Universidad Autónoma de Madrid
Escuela Politécnica Superior

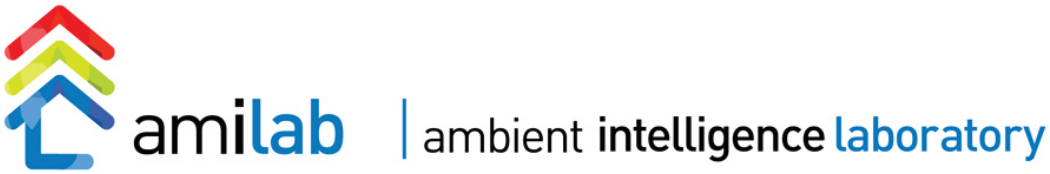
SISTEMA ADAPTATIVO PARA LA ASISTENCIA DE PERSONAS CON NECESIDADES ESPECIALES EN SUS TAREAS DE LA VIDA DIARIA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Autor: Juan Carlos Torrado Vidal
Tutores: Javier Gómez Escribano - Germán Montoro Manrique

Junio 2013



Resumen y palabras clave

Resumen

Hoy en día muchas personas con algún tipo de discapacidad cognitiva requieren asistencia para realizar tareas cotidianas como preparar un café o lavar la ropa correctamente. Las nuevas tecnologías suponen un punto de apoyo para la mejora de nuevas formas de asistencia, que contribuyan ayudando tanto en la propia realización de la tarea como en la adquisición progresiva de independencia a la hora de realizarlas.

Partiendo de un trabajo previo, este Trabajo de Fin de Grado consiste en la adaptación y extensión de un sistema basado en teléfonos móviles y códigos QR a nuevas necesidades, para abarcar más tipos de usuarios a los que asistir de manera completa, así como facilitar la edición y configuración de este sistema a los tutores de dichos usuarios.

Palabras clave

Necesidades especiales, discapacidad cognitiva, diversidad funcional, aplicaciones móviles, códigos QR, Inteligencia Ambiental, manual adaptado, Computación Ubicua.

Abstract and Keywords

Abstract

Nowadays people with mental impairment use to have difficulties on their daily life activities engagement. Normally, they learn how to do them through the direct assistance of educators and specialized people.

Due to this problem is why AssisT-Task was born. It is a project built upon a previous work based on assistive technologies using QR codes and mobile devices. AssisT-Task is meant to help these people on doing their daily life activities and gradually gaining autonomy through its use. This application generates step-by-step manuals that can be adapted to the circumstances and needs of the user such as support for a wider set of tasks, enabling user interaction during the use of the application.

Keywords

Special needs, mental impairmente, mobile applications, QR codes, Ambient Intelligence, adapted manuals, Ubiquous Computation

Agradecimientos

En primer lugar, muchas gracias a mis tutores Germán y Javi, por todo lo que me han ayudado y enseñado, así como la gran acogida que hemos tenido mis compañeros y yo en el AmiLab, donde, finalmente, me he sentido como en casa.

Con este trabajo termino una etapa muy importante e inovidable en mi vida, en el aspecto académico y profesional. Esto no hubiera podido ser posible sin toda la gente que ha estado a mi alrededor y ha hecho única esta experiencia.

Por una parte, quiero nombrar a mis compañeros de carrera, con los que tantísimas horas he pasado. Pese a lo dura que ha sido la carrera y lo mal que lo hemos pasado con nuestras "entrañables" prácticas, sólo recuerdo risas y buenos ratos. He conocido a muchísima gente y verdaderos amigos. Todos ellos deberían quedar aquí nombrados, pero especialmente Adal y Santi, que me han acompañdo durante toda la carrera, en lo profesional y lo personal. Esos ratos en el césped después de comer enseñan más que un libro de programación, os lo aseguro.

Gracias a mis amigos, que siempre han estado ahí. Han sido un apoyo constante y una fuente inagotable de alegrías y risas.

Y por último, y no menos importante, a mis padres y mi hermana, que me han apoyado durante toda mi vida y siempre han estado y estarán ahí para cuando lo necesite. Muchas gracias, de verdad.

Quién sabe lo que vendrá con esta nueva etapa que comienzo. Sin embargo, tengo claro que habrá siempre gente con la que completar este apartado, quedándome corto.

Juan Carlos Torrado Vidal
Junio de 2013

Índice

Índice de Figuras

Índice de Tablas

1 Introducción

1.1. Motivación

Las tecnologías para la asistencia son aquellas que facilitan la ejecución de actividades o la interacción con el entorno físico y social de las personas en situación de discapacidad [1]. Corresponden a todo tipo de objetos, equipos, sistemas, máquinas, herramientas, productos, instrumentos, programas y/o servicios que pueden ser utilizados para aumentar, mantener, compensar, mejorar o reemplazar las capacidades funcionales de estas personas. Dentro de las tecnologías para la asistencia, este trabajo se enmarca de las denominadas "de alta tecnología", que comprenden todo aquello relacionado con las Tecnologías de la Información (TIC).

La labor planteada surge de la necesidad de desarrollar una herramienta de asistencia enfocada a un conjunto de usuarios con una diversidad funcional mayor que la planteada inicialmente en el trabajo previo de Gómez *et al.* [11], si bien se aprovechará la funcionalidad de la que se disponía para dar cabida a estos usuarios. Es necesario, por tanto, un nuevo planteamiento, que lleva a reconsiderar el diseño tanto de la aplicación móvil que utilizará el usuario, como de la arquitectura en la que se apoya el sistema desarrollado.

Este Trabajo de Fin de Grado (TFG) se ha implementado en el Laboratorio de Inteligencia Ambiental (AmILab), y manteniendo contacto con expertos de la Funda-

[1] Assistive Technology Act, 1988

ción de Síndrome de Down de Madrid [2], que han aportado directrices en cuanto a la usabilidad y accesibilidad máxima de la interfaz de usuario.

1.2. Objetivos

El objetivo principal de este TFG es estudiar nuevas estrategias de apoyo y rehabilitación de personas con necesidades especiales mediante el uso de nuevas tecnologías. Concretamente, se trata de una aplicación en dispositivo móvil que ayudará al usuario a realizar tareas de la vida diaria, ofreciendo manuales adaptados a sus necesidades. El sistema deberá ser lo suficientemente flexible como para poder asistir en tareas que se adapten a las necesidades de un usuario en cada momento, ofreciendo una cierta interacción con el mismo. Actualmente, esta labor es realizada por los expertos que asisten a estas personas. Sin embargo, supone un coste logístico y económico que se puede eliminar con el uso de este sistema, posibilitando que un educador asista a más usuarios y sin necesidad de una intervención directa e intrusiva en la vida diaria del individuo.

1.3. Visión general

Como se ha comentado en secciones anteriores, las tecnologías para la asistencia abren nuevas posibilidades para introducir los beneficios de las TIC en el ámbito del apoyo a personas con discapacidad cognitiva. En lo que se refiere a este trabajo, se aprovecha la popularidad que tienen hoy en día los dispositivos móviles. Ya se han convertido en un elemento que forma parte de la vida diaria de cualquier persona, sobre todo desde que son capaces de ofrecer servicios como Internet, multimedia y acceso a redes sociales.

Es por eso que los dispositivos móviles, que disfrutan de esta versatilidad, entrañan un potencial muy interesante en relación con la ayuda a personas con necesidades especiales. En concreto, este trabajo se centra en personas con discapacidad cognitiva, cuyas características se exponen en la Sección 2. Incluso para muchas de estas personas el móvil es un elemento de uso cotidiano, y se desenvuelven con mayor o menor soltura durante su uso. Uno de los inconvenientes más problemáticos a la hora de insertar un elemento tecnológico en la vida de una persona con necesidades especiales en su vida cotidiana es el aprendizaje y entrenamiento previo que se requiere, aunque sea con el objetivo de mejorar su autonomía en el futuro. Escogiendo los dispositivos móviles actuales como plataforma del sistema de asistencia, se pretende que para el usuario

[2] www.downmadrid.org

esta parte de familiarización sea más suave y natural que con cualquier otro dispositivo tecnológico especializado.

1.4. Estructura del Trabajo Fin de Grado

Este Trabajo Fin de Grado está organizado de la siguiente manera:

- El Capítulo 2 trata sobre las personas con discapacidad cognitiva. Se hace una clasificación sobre los distintos tipos y grados de discapacidad que existen, así como el abanico de necesidades que presentan.
- El Capítulo 3 hace una revisión del Estado del Arte relacionado con las tecnologías desarrolladas con un objetivo relevante a la motivación de este trabajo.
- El Capítulo 4 es una descripción del sistema del que consta este trabajo, de la arquitectura que lo soporta, la funcionalidad que ofrece y ejemplos explicativos.
- El Capítulo 5 describe un ejemplo completo de actividad de la vida diaria que se ha implementado para este sistema.
- El Capítulo 6 es una recopilación de conclusiones sobre el trabajo realizado y un esbozo de las líneas posibles de trabajo futuro.
- En los anexos se encuentra código perteneciente a los ejemplos descritos y a elementos de la ontología programada para este sistema

2 Personas con necesidades especiales

Cuando nos referimos en el texto a personas con necesidades especiales, estamos haciendo referencia a casos de **discapacidad intelectual**.

2.1. ¿Qué es la discapacidad intelectual?

Según la AAMR (American Association for Mental Retardation)[1], *"La discapacidad intelectual se refiere a limitaciones sustanciales en el funcionamiento intelectual. Se ha de manifestar durante el desarrollo, antes de los 18 años de edad"*.

El tipo de discapacidad intelectual de una persona debe enmarcarse definiendo las limitaciones que la caracterizan. Estas limitaciones deben considerarse en un contexto típico de sus iguales en edad y cultura, así como tener en cuenta la diversidad lingüística y cultural que pueda existir. Pueden coexistir con capacidades, y siempre deben ser definidas describiendo el perfil de apoyo que necesitan.

Se dice que la definición de la discapacidad intelectual es **multidimensional** por poder afectar de distinta manera y grado a distintos aspectos de la vida del individuo:

- Dimensión I: Habilidades intelectuales
- Dimensión II: Conducta adaptativa (conceptual, social y práctica)
- Dimensión III: Participación, interacciones y roles sociales

[1] http://www.aaidd.org/

- Dimensión IV: Salud (salud física, salud mental)
- Dimensión V: Contexto (ambientes y cultura)

2.2. Tipos de discapacidad intelectual

Si bien es cierto que la discapacidad intelectual es un fenómeno complejo y que afecta, como se ha visto, a distintos aspectos y de diversas maneras, los distintos grados de discapacidad vienen dados únicamente por los resultados de CI (cociente intelectual) obtenidos al someter al individuo a los *Test de Inteligencia Weschler* [20]:

- CI 50-69: Discapacidad intelectual leve
- CI 35-49: Discapacidad intelectual moderada
- CI 20-34: Discapacidad intelectual grave
- CI <20: Discapacidad intelectual profunda

En algunos casos, dependiendo de la causa que originó la discapacidad, el individuo puede presentar además problemas de movilidad, que dificultan aún más su autonomía. Este trabajo se mueve en las regiones entre el grado leve y moderado de discapacidad intelectual. Siempre habrá que tener en cuenta particularidades de los individuos en cuanto a la necesidad de usar la aplicación.

2.3. ¿Qué aporta el sistema propuesto?

En cuanto a la discapacidad intelectual, este trabajo pretende proporcionar un apoyo a la hora de enfrentar las dificultades que plantea el desarrollo de tareas secuenciales de mayor o menor nivel de complejidad. Estas tareas serán aquellas que el usuario debe terminar realizando por sí mismo si se desea realizar un progreso en cuanto a autonomía y responsabilidad. La intención es ambiciosa: se pretende que, además de aprender a realizarla tarea en sí que se le plantee, adquiera una adaptación progresiva a la realización en general de tareas, de forma ordenada, coherente y segura. Este planteamiento garantiza que el usuario, con el tiempo, pueda adquirir habilidades no sólo útiles para su autonomía en el día a día, ya que en el mismo ámbito laboral este tipo de destrezas son necesarias para una inserción adecuada y progresiva.

Este trabajo hace hincapié en las **actividades de la vida cotidiana**, aunque también está relacionado con las habilidades de **cognición** y **comunicación**.

Las actividades de la vida diaria requieren de unas habilidades específicas a la hora de planificar o seguir pasos de una serie, y se consideran de especial interés dado el nivel de independencia que aportan al paciente. Es, por tanto, interesante enfocar este trabajo como un método diferente de rehabilitación. Y, más concretamente, en aquel tipo de rehabilitación que tiene por objetivo alcanzar en el paciente la mayor capacidad física, funcional y social. El sistema que propone este trabajo sería situado en la fase ambulatoria de dicha rehabilitación. Normalmente, ésta se realiza con un terapeuta, que a lo largo de diferentes sesiones planifica unas actividades, las explica al paciente y le acompaña durante la realización de las mismas, asisitiéndole cuando es necesario. Con este sistema, el terapeuta pasaría a segundo plano, siendo necesario únicamente en casos de peligro y en la evaluación del progreso del paciente.

Puede resultar interesante que el individuo, además, se habitúe al uso de tecnologías como los dispositivos móviles, no sólo por el uso de este sistema, sino por las facilidades que les pueden suponer en otros ámbitos de su vida, al ser algo tan extendido en la sociedad y de uso más que habitual. Por otra parte, esto también supone que los usuarios deberán cumplir una serie de requisitos impuestos por las dificultades de comunicación que puedan tener: hay usuarios que pueden no saben leer, otros con problemas para asimilar instrucciones habladas; en cuyos casos se deberá prestar atención al tipo de refuerzo (visual, acústico) que sea adecuado para una asistencia integrada.

3 Estado del Arte

Las tecnologías para la asistencia vienen definidas en el estándar ISO 9999 [18]: "*aquellos productos, instrumentos, equipos o sistemas técnicos fabricados expresamente para ser utilizados por personas con discapacidad y/o mayores; disponibles en el mercado para prevenir, compensar, mitigar o neutralizar una diversidad*".

Esta definición engloba muchos elementos, clasificados por la Norma a lo largo de una amplia numeración que abarca desde las prótesis físicas que ayudan al movimiento hasta las gafas para suplir defectos visuales. Este TFG se centra en las clasificadas bajo el código *22.33.06*, que comprende las tecnologías para la asistencia de tipo sistema de comunicación móvil (ordenadores portátiles, PDAs, teléfonos móviles...).

El campo de estudio de las tecnologías para la asistencia de este tipo es muy amplio, si bien no existen demasiados trabajos centrados en el desarrollo de estas tecnologías para personas con discapacidad cognitiva. LoPresti *et al.* [16] aportan una revisión del estado del arte de las tecnologías para la asistencia de estas personas. Además, se enumeran una serie de requisitos necesarios que deben reunir estas tecnologías para adaptarse a esta discapacidad, como al presentación de información en diversos modos (gráficos, vídeo, audio, etc.) y de forma más simple posible, de forma secuencial y atendiendo a otras limitaciones que pueden acompañar a la discapacidad cognitiva en algunos casos (limitaciones físicas, sensioriales, etc.). Es por estas razones que se sugiere que se utilicen metodologías centradas en usuario, así como incluirle en el proceso de diseño. La clasificación que se hace en este artículos sobre los distintos trabajos estudiados se basaba en el área cognitiva que pretenden mejorar:

- Tecnologías para la memoria, planificación y resolución de problemas y dispositivos ortopédicos sensibles al contexto.

    - Tecnologías compensatorias para la memoria
    - Tecnologías compensatorias para planificación y resolución de problemas
    - Dispositivos ortopédicos cognitivos sensibles al contexto
- Tecnologías para discapacidades en el procesamiento sensorial y asuntos sociales y de comportamiento.
    - Tecnologías compensatorias para el procesado sensorial
    - Tecnologías para asuntos sociales y de comportamiento

Por otro lado, Braddock *et al.* presentan en [2] una revisión del estado del arte de las TAC basado en áreas tecnológicas. Proponen una clasificación en dos grupos:

- Tecnologías para el apoyo personal, como son las PDAs, comunicación y aprendizaje asistidos por ordenador y Diseño Universal
- Sistemas de cuidado asistido. Entre ellos encontramos casas y sistemas de transporte inteligentes, robots personales y tecnologías de realidad virtual.

Carmien *et al.* en [5] también describen la necesidad de desarrollar software específico para estas personas, en sustitución del trabajo de tutores y educadores, pero sin excluirles. En su planteamiento, estos profesionales toman un papel menos presencial, colaborando en la configuración del software atendiendo a las necesidades concretas de cada usuario, ya que son ellos los encargados de valorarlas y establecer las pautas de apoyo que necesitan. Asímismo, se describe la idea de diseñar software que proporcione un apoyo orientado para "dejar de ser utilizado", esto es, que ayude al usuario a realizar esas actividades para las que el software le apoya sin ese mismo software, en un futuro.

En [13], un proyecto español, Hidalgo *et al.* plantean relegar a los educadores a un papel aún más pasivo, proponiendo un software desarrollado (*ATHENA*) que aprovecha los registros de salida de su software de generación de manuales interactivos para elaborar nuevas planificaciones basadas en el aprendizaje automático que hace su sistema. De esta manera, la actuación de un tutor es meramente supervisoria y de carácter periódico.

Una revisión más actualizada del estado del arte, y centrada en los sistemas de accesibilidad móvil y de acceso a la información en cualquier momento y lugar, la hacen Tsui y Yanco en [19]. Este trabajo se centra en dispositivos para la ayuda de realización de tareas secuenciales, muy relacionado con el trabajo que se realiza en este TFG. Clasifican los trabajos de este tipo en tres grupos: los que no utilizan recursos

tecnológicos, los que hacen uso de dispositivos electrónicos personales, y los que utilizan inteligencia artificial. Este TFG atañe al segundo grupo de los mencionados.

Ya centrados en los sistemas de apoyo a la realización de tareas basados en dispositivos electrónicos personales, encontramos que ya se distinguen aquellos que sólo soportan tareas lineales de los que permiten bifurcaciones o saltos. Del primer tipo tenemos iPACS [1] y iPrompt [12], diseñados para productos de Apple (iPhone, iPod Touch y iPad). Son dos sistemas que presentan una colección de imágenes secuenciadas, acompañadas de notas de voz o etiquetas, para ayudar a la realización de tareas. iPacs incluye herramienta de autor para los educadores, e iPrompt, opciones de configuración para establecer *timeouts*.

Ejemplos de sistemas que permiten bifurcaciones y saltos son PEAT (*Planning and Execution Assistant and Trainer*) [15], VICAID [9], Virtual Assistant [7] y MAPS (*Memory Aiding Prompting System*) [3]. Este último es muy interesante, ya que propone un sistema de mensajes diseñado para ayudar a personas con discapacidad cognitiva en su vida diaria mediante la descripción de éstas en guiones compuestos por imágenes y textos. El sistema está compuesto por la interfaz de usuario, diseñada pensando en sus necesidades y con adaptación a sus restricciones y a las del dispositivo; y la herramienta de autor, que permite a los educadores describir los pasos para construir tareas de forma sencilla.

Por último, Mechling en [17], analizó profundamente el carácter de las indicaciones que se pueden utilizar, restringidas a la plataforma que se utilice para la asistencia, y al tipo de discapacidad que sufra el usuario en cada caso. Además, hace especial hincapié en la utilidad de la combinación de estímulos visuales, auditivos y vibratorios, especialmente orientado a plataformas móviles y de uso común para el usuario. Sin embargo, el interés de su aporte radica en las pruebas que realiza con usuarios, en las que lleva a cabo una evaluación segmentada en:

- Si el usuario completa la tarea,
- Concentración y dedicación del usuario en su elaboración
- Comportamiento del usuario durante la tarea
- Precisión en la realización
- Dificultad al iniciarlas
- Transición entre tareas
- Fluidez durante el proceso

De la misma forma, Chang et al. en [6] señalan los principales problemas que tiene un individuo con discapacidad cognitiva para realizar sus tareas cotidianas. Además del análisis secuencial semejante al que realizó Carmien en [4], en este caso se observa especialmente la dificultad que les supone recordar la mecánica concreta de una rutina. Los autores concluyen justificando la necesidad de realizar planificaciones personalizadas, ya que el carácter de las indicaciones son críticos a la hora de que el usuario tolere adecuadamente el guiado. En este caso, la evaluación se realiza con PDAs y balizas Bluetooth.

3.1. Conclusiones

En este capítulo se han analizado, por una parte, los distintos proyectos existentes para la ayuda en la realización de tareas cotidianas a través de sistemas electrónicos. Por otro lado, se han resumido las principales conclusiones que los distintos autores extraen de sus evaluaciones y el uso de los sistemas desarrollados.

Algunos de los sistemas qeu se han visto, se basan en dispositivos móviles. Se da importancia de esta manera a la idea del acceso a la información en cualquier momento y lugar, mediante un objeto que el usuario lleva consigo.

Los trabajos desarrollados para iOS (iPACS e iPrompt) y PDA (MAPS) ofrecen interfaces enriquecidas pero quizá algo complejas y de accesibilidad limitada para algunos usuarios con dicapacidad cognitiva (por ejemplo, usuarios que no puedan leer o tengan problemas visuales no podrían usar la aplicación por sí mismos). Tener esto en cuenta a la hora de construir una interfaz más accesible es muy interesante, al igual que la manera en la que se muestra la información, a saber: imágenes para usuarios que no sepan leer o instrucciones por voz para usuarios con problemas visuales.

La posibilidad de construir tareas no lineales en PEAT, VICAID y Virtual Assistant resulta de especial interés para este trabajo, para poder aaptar el manual que se genera a la situación en la que se encuentra el usuario.

En este trabajo se implementará una herramienta basada en sistema móvil, en la que se muestren secuencias de dibujos o fotografías acompañadas de texto, en dispositivos móviles. Se incluirá la posibilidad de implementar tareas no lineales, con repeticiones de tareas y bifurcaciones. El fin último de esta aplicación es ayudar al usuario para que adquiera las destrezas necesarias que le permitan realizar tareas de la vida cotidiana sin la asistencia de un tutor ni una herramienta como la que le proponemos, pero hacerlo mediente su uso.

4 AssisT-Task

4.1. Propuesta

Queremos diseñar un sistema basado en aplicación móvil que nos permita escanear un código con el dispositivo y generar manuales paso a paso para realizar una actividad de la vida diaria descrita en dicho código.

A la hora de efectuar el diseño del sistema que daría base a este trabajo, se tuvo en cuenta que se debía seguir un modelo de diseño que tuviese muy en cuenta la interacción con el usuario final. Como en este caso, el usuario final no está capacitado para aportar retroalimentación durante las distintas fases o iteraciones de desarrollo, se mantuvo contacto con expertos de la FSDM (Fundación de Síndrome de Down de Madrid)[1] para incluir al usuario en el proceso.

Actualmente, existe una tendencia creciente en la utilización de modelos centrados en el usuario, que se caracterizan por fases más cortas, más retroalimentadas y donde el usuario juega un papel fundamental antes, durante y después de la construcción del software.

El Diseño Centrado en el Usuario (DCU) es una filosofía de diseño que tiene por objetivo la creación de productos que resuelvan necesidades concretas de sus usuarios finales, consiguiendo la mayor satisfacción y mejor experiencia de uso posible con el mínimo esfuerzo de su parte. El objetivo principal es el aseguramiento de la usabilidad y accesibilidad del sistema construido.

[1] http://www.downmadrid.org/

4.2. Requisitos del sistema

El estándar *ISO 13407* [14] constituye un marco guía para conseguir el desarrollo de sistemas interactivos usables incorporando el DCU durante el ciclo de vida del desarrollo. Especifica unas pautas dirigidas a entender el desarrollo de un sistema de acuerdo a cuatro principios:

Entender y especificar el contexto de uso Como ya se ha comentado en el capítulo 2, la aplicación desarrollada está enfocada para sustituir la clásica sesión de rehabilitación en la que el paciente realiza actividades cotidianas de mayor o menor complejidad mientras es observado por su tutor. El contexto de uso, por tanto, está entendido como el ámbito cotidiano del usuario, es decir, el uso de la aplicación pretende estar integrado en su actividad diaria, si bien es cierto que el fin último consiste en que el paciente obtenga unas destrezas mediante su uso que le permitan realizar las tareas diarias sin el soporte y guiado que proporciona la aplicación.

Especificar los requisitos de los usuarios Trasladado al contexto de este trabajo, tratar los requisitos de los usuarios consiste en algo diferente. En un sistema desarrollado mediante una metodología DCU estándar, las opiniones que marcan las pautas de desarrollo son las de los usuarios del sistema a desarrollar. Sin embargo, en este caso el usuario final no puede aportar tanto a los requisitos como sus tutores, que son más conscientes de sus necesidades. Este papel ha sido llevado a cabo en este trabajo por los expertos de la FSDM. Más adelante se realizará un desglose de las pautas que inicialmente se establecen para hacer de la aplicación un sistema usable y accesible.

Diálogo simple y natural El estándar marca esta pauta genérica, mientras que, tratándose de personas con diversidad funcional cognitiva, no sólo se deberá cuidar el diálogo sino cualquier estímulo que la aplicación genere hacia el usuario.

Producción de soluciones de diseño Entendida como la confección y planificación de la interfaz a utilizar, así como las opciones de configuración de las que se deberá disponer y la arquitectura que lo soporta.

Estas pautas pueden entenderse como pasos a realizar de forma iterativa, a modo de un ciclo clásico de desarrollo análisis-diseño-codificación-pruebas. El ISO 13407 describe los principios básicos sin estipular métodos específicos. La secuencia de realización o seguimiento de las actividades y el nivel de esfuerzo y detalle aproximado a cada proyecto varía dependiendo del entorno de diseño y el estado de proceso del mismo.

Los requisitos principales que se extraen para cumplir con los objetivos especificados en el capítulo 1.2 son los que se especifican a continuación:

Accesibilidad móvil Se tratará de un sistema que el usuario utilizará desde su dispositivo móvil, en cualquier sitio y momento que desee, siempre y cuando tenga conexión a la red. El diseño deberá tener en cuenta la variedad de dispositivos que existen en el mercado, intentando que la aplicación funcione en la mayor variedad de dispositivos posible. En el dispositivo móvil se deberá presentar toda la información necesaria para que el usuario sepa completar la tarea, sin tener que desplazarse a algún sitio para obtener la información: le bastará con las instrucciones que le proporciona el dispositivo que lleva consigo.

Presentación de la información Teniendo en cuenta los distintos problemas que pueden presentar personas con discapacidad cognitiva a la hora de leer, visualizar o escuchar información, la aplicación deberá dar soporte a que la información se pueda transmitir a los usuario de forma textual, gráfica o por voz.

Simplicidad Se deben eliminar todos aquellos elementos de la interfaz del usuario que no sean estrictamente necesarios para la resolución de la tarea y puedan desviar la atención del mismo. Se deben tener en cuenta las notificaciones que produce el dispositivo móvil, para adaptarlas a este tipo de usuarios.

Modelo centrado en el usuario Se deberá incluir al usuario en el proceso de diseño. En este trabajo, este requisito se satisface siguindo las pautas extraídas de las auditorías realizadas con los expertos de la FSDM.

4.3. Arquitectura propuesta

Como se ha especificado en los requisitos, es necesario que se utilice un dispositivo capaz de acceder a la información en cualquier momento y lugar (acceso *ubicuo*). Hoy en día, esto es realizable a través de ordenadores portátiles, sin embargo la idea es que el usuario utilice un dispositivo que lleve consigo y utilice de manera sencilla. Los teléfonos móviles actuales son dispositivos muy extendidos en la sociedad, que se manejan bien y tienen una duración mayor de batería que un ordenador portátil. Además, se trata de un sector en auge, en el que los dispositivos que salen al mercado y la gente va adquiriendo son cada vez más potentes y tienen más funcionalidad: se mejora el hardware (procesadores y memorias mejores, cámaras de alta resolución, sistemas de localización, etc.), el software (sistemas operativos como Android, iOS y Windows Phone 7) y las posibilidades de comunicación que ofrecen (redes 3G y Wi-Fi).

En cada vez más empresas y proyectos, los sistemas diseñados se enfocan para tener su versión móvil antes que el programa en ordenador. En este caso, que además el dispositivo móvil se ajusta a los requisitos establecidos, esta filosofía hace que se parta con la ventaja de aportar algo en un sector muy extendido en la sociedad: un valor añadido para este trabajo.

Sin embago, aunque el acceso a la información deba ser posible en cualquier momento y lugar, no significa que la aplicación del móvil deba contener toda la información, como se podría pensar en primer lugar. De ser así, se dificultaría la labor de los educadores y tutores de estas personas, que deberían acceder a los móviles de todos los alumnos cada vez que quisiesen hacer un cambio (añadir, modificar o suprimir la información de alguna actividad). Es por esto que la solución idónea consiste en centralizar la información en un componente al que accedan las terminales móviles mediante conexión a Internet: una arquitectura cliente-servidor.

De esta manera, los educadores sólo tendrían que acceder a esta información almacenada en el servidor si quisiesen realizar alguna modificación, y ese cambio sería efectivo para todas las terminales que accediesen a ese servidor en la próxima ejecución, en lugar de tener que acceder a todas las terminales con la aplicación en uso y actualizarlas. Este servidor se encargará también de gestionar la información eventual sobre decisiones que toma el usuario al respecto de tareas no lineales, es decir, considera el contexto y necesidades del usuario.

El usuario, por su parte, recibe a través de la interfaz la instrucción que necesita para completar la tarea sin necesidad de desplazarse para obtenerla. De esta manera evitamos la distracción que puede suponer para un usuario con discapacidad cognitiva un desplazamiento.

La arquitectura general del sistema se puede ver en la figura 4.1.

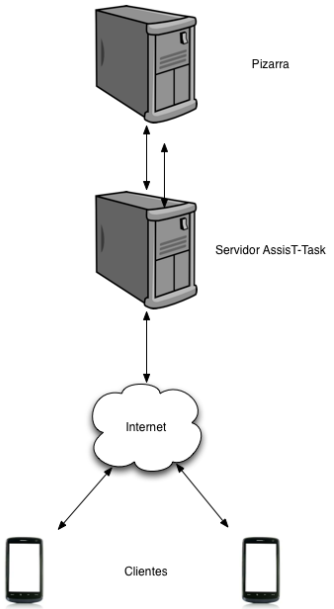


Figura 4.1: Esquema general de la arquitectura

Se utilizará un paradigma orientado a objetos, en concreto implementación en len-

guaje Java para cliente y servidor (en el cliente, se utilizará Android, pero a efectos de lenguaje, se trata también de Java).

Las comunicaciones entre los distintos elementos de la arquitectura se resolverán utilizando ICE (Internet Communications Engine) [2].

4.3.1. ICE (Internet Communications Engine)

Cuando se plantea un sistema distribuido, el diseño de las comunicaciones entre los elementos que lo conforman es una cuestión importante. Sin embargo, existen herramientas como ICE que resuelven la implementación a bajo nivel de las comunicaciones, permitiendo centrarnos en la lógica de la aplicación. ICE ofrece un lenguaje propio (*slice*) en el que definir modelos de datos. A partir de estos modelos descritos, ICE genera código en un amplio abanico de lenguajes que permite que esos datos puedan ser usados en red.

Tanto los modelos de datos como los elementos de la estructura (todos ellos, objetos Java), han sido definidos en *slice*, generando a partir de esa reperentación unas clases que servirán para que los módulos que necesiten acceder a esos objetos en red puedan obtener referencias a los mismos y trabajar sobre ellos. De este modo queda completamente transparente al programador toto lo referente a comunicaciones.

De ahora en adelante, todas las peticiones y comunicaciones que se indiquen, al estilo 'èl servidor le envía al cliente las tareas", se traducirían, por tanto, en "el cliente obtiene las referencias ICE a los objetos del servidor que desea consultar o modificar". Sin embargo, se optará por la expresión sencilla, por simplicidad.

4.4. El servidor

La implementación del trabajo presentado en este TFG se ha llevado a cabo en el Laboratorio de Inteligencia Ambiental AmILab. En este laboratorio se ha venido desarrollando una capa intermedia o *middleware* para entornos inteligentes soportada por ontologías [10]. Este sistema es el llamado "*The Blackboard*" (la Pizarra), ya que está basado en la metáfora de la pizarra [8].

En la Pizarra se modelan objetos del entorno, cuya estructura se describe en un esquema. Este esquema contiene la descripción del entorno, en términso de clases, propiedades, capacidades y las relaciones que hubiera entre ellas. A esto se le llama

[2] http://www.zeroc.com/ice.html

"modelo ontológico". Los objetos que se modelan de acuerdo a estos esquemas, a modo de instanciaciones, se conocen como entidades, y se almacenan en los repositorios de la Pizarra de manera que son accesibles desde al estructura global de la misma. Estas entidades pueden representar:

- **Objetos físicos:** sensores, ordenadores, personas, etc.
- **Objetos virtuales:** imágenes, información personal, etc.

De esta manera, para el diseño se ha optado por modelar las actividades de la vida diaria como entidades de la Pizarra (se trata de entidades que representan "objetos virtuales" del mundo que atañe a este proyecto). De esta manera, la información relativa a las tareas quedará centralizada en la Pizarra y estará disponible para los educadores cuando la quieran actualizar.

4.4.1. Estructura del servidor

En la figura 4.2 se puede ver un esquema más detallado de la parte servidor de la aplicación. Se compone de los siguientes elementos:

1. La Pizarra: tiene un módulo que gestiona el Esquema, esto es, la ontología del entorno que se haya modelado. En nuestro caso, se trata de la definición de las estructuras sobre las que se definirán las tareas que vaya a realizar el usuario a través de AssisT-Task, a saber: Action, Routine, Task y Fragment(ver apartado 4.4.2). La estructura de estos elementos es descrita más adelante. Por otra parte, el Repositorio se encarga de almacenar y gestionar el acceso a las entidades que se hayan descrito mediante la ontología definida en el Esquema. Estas entidades habrán sido almacenadas a partir de su definición en un fichero externo, al igual que en el Esquema con la ontología. Los otros módulos de la pizarra tienen otras funciones que no resultan de especial relevancia para este trabajo: *drivers*, lógica de autenticación, elementos de comunicación entre módulos y APIs, etc.
2. El servidor intermedio entre la Pizarra y la aplicación móvil. Se compone de los siguientes módulos:
   - **Modelo actividades**: módulo que se comunica con el Repositorio de la Pizarra para obtener los datos de las entidades e inicializar un modelo propio (simplificado y sin elementos propios de la Pizarra) de Actions, Routines, Tasks y Fragments. Este modelo utiliza los elementos autogenerados por ICE para permitir que esos objetos puedan ser accesibles en red. De esta

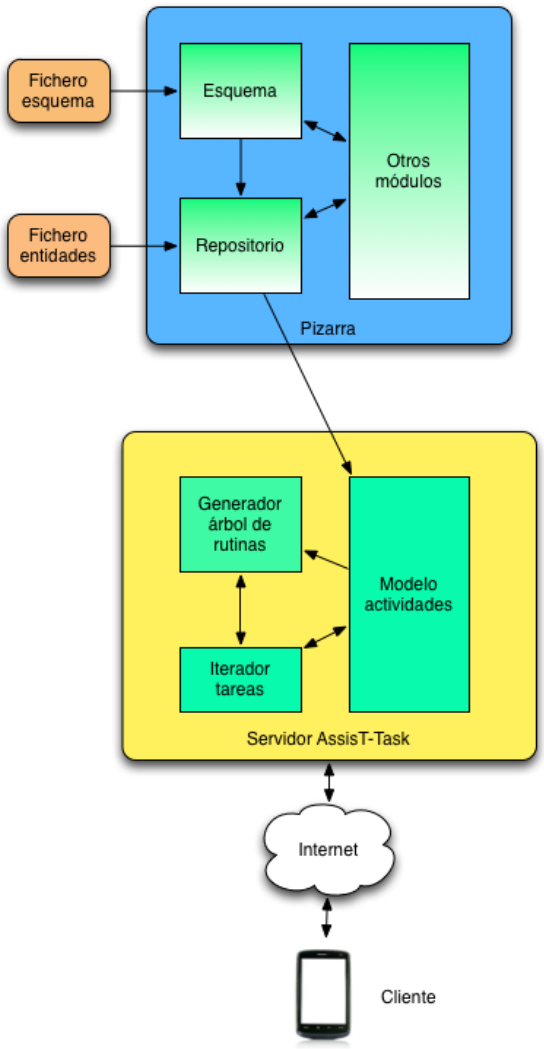


Figura 4.2: Esquema de la arquitectura del servidor

forma, el dispositivo móvil podrá manejarlos y obtener sus datos aunque estén en el servidor.

- **Generador árbol de rutinas**: con los datos del modelo de actividades, se construye el árbol que representa la secuencia de tareas que se van a ejecutar en el dispositivo móvil del usuario. Para ello, analiza todas las relaciones entre los objetos del modelo y establece un orden en el cual los elementos deberán ser recorridos. Esta acción es la primera que se realizará cuando el cliente solicite iniciar una actividad: se realizarán peticiones a la Pizarra para obtener todos los objetos de los que consta la actividad que se ha escaneado, se almacenan en el servidor y se ordenan en la manera en la que se recorrerán a medida que el cliente los vaya solicitando (esto es, a medida que el usuario va avanzando en la tarea y va solicitando los pasos siguientes

a realizar).

- **Iterador sobre tareas**: se trata de la parte del servidor que recorre el árbol de rutinas según los criterios mencionados en el ejemplo anterior y atendiendo a los datos de las actividades que hay en el modelo. También se encarga de gestionar las peticiones que recibe del dispositivo móvil, teniendo en cuenta que las tareas que alteren el orden normal establecido por el árbol de rutinas construido supone que las peticiones y respuestas que se llevarán a cabo son distintas. Por ejemplo, cuando al servidor se le solicite la siguiente tarea, y ésta lleve repeticiones, esta respuesta del servidor será especial, porque además de la tarea llevará adjunta la cifra que indica cuántas veces se deberá repetir. Si, por otra parte, se trata de una tarea de selección múltiple, en lugar de la cifra de repeticiones, el servidor deberá enviar en la respuesta al cliente la lista de opciones de selección perteneciente a la tarea.

En el siguiente apartado veremos qué estructura se ha escogido para modelar las actividades.

4.4.2. Descripción de las actividades

Las actividades se describen en términos de "Routines" (rutinas, que comprenden una actividad completa), "Tasks" (tareas, que son los pasos que componen las rutinas) y "Fragments" (fragmentos de información acerca de una tarea en forma de texto o imagen). De esta manera, se puede representar una actividad en forma de árbol.

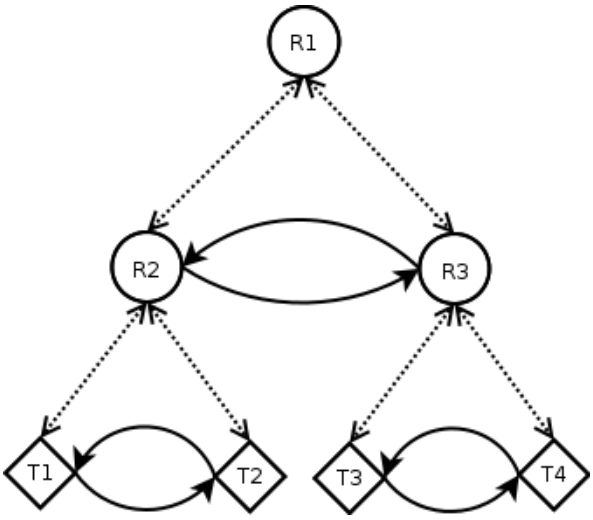


Figura 4.3: Ejemplo de árbol de rutinas

En la figura 4.3 se pede observar un ejemplo de esta representación en forma de árbol de una rutina. Los nodos son elementos Routine, Task o Fragment, y las ramas

que los unen son las relaciones entre ellos. En este caso, la rutina R1 está formada por dos subrutinas (R2 y R3), que, a su vez, comprenden las tareas T1 y T2, y T3 y T4, respectivamente. El árbol completo constaría, a su vez de los Fragments que componen las tareas, pero en el ejemplo se han omitido para simplificar la explicación, ya que no son de mayor relevancia para explicar el ejemplo. Las líneas discontinuas son de parentesco (nodos padre-hijo) y las continuas son de secuenciación (paso anterior y paso siguiente).

La aplicación, en primer lugar, busca la primera tarea ejecutable. Para ello, realiza una búsqueda en profundidad hasta encontrar la primera tarea, ejecutando las sucesivas secuencialmente, en el orden en el que están dispuestas. Por ejemplo, en la figura 4.3 la aplicación haría un recorrido R1-R2-T1, obteniendo la primera tarea de la secuencia. Dicha rutina R2 contiene dos tareas, que serán recorridas en el orden T1-T2. A continuación se procedería de la siguiente manera para recorrer el árbol:

1. Comprobar si T2 tiene tarea siguiente: no la tiene.
2. Comprobar si T2 tiene nodo padre: Sí tiene, nodo R2.
3. Comprobar si R2 tiene una relación "siguiente": Sí tiene, con R3.
4. Obtener la primera tarea de R3:
    a) Encontrar todas las tareas que tienen una relación "está compuesto de" con R3.
    b) Buscar la que no tenga relación "anterior" con nadie (la primera de la subrutina): T3.
5. Por tanto, la siguiente a T2 es T3

Recorriendo todo el árbol con este criterio y haciendo esas preguntas a la Pizarra, obtendríamos que la secuencia de tareas que percibiría el usuario sobre esta descripción es T1-T2-T3-T4.

Estas "preguntas" son conexiones que tendríamos que establecer con la Pizarra. Se puede deducir con este ejemplo que se produce una carga de tráfico de conexiones con la Pizarra bastante alto (seis conexiones para avanzar un sólo paso).

Es por esto que se decidió incluir un servidor intermedio entre la Pizarra y el cliente. De esta manera, dicho servidor obtiene de la Pizarra todas las entidades Routine que componen una actividad que ha escaneado el cliente con el móvil, construye el árbol y lo almacena en memoria hasta que finaliza la aplicación por parte del usuario. De esta

manera, toda esa serie de preguntas se reducen a una sola: comprobar cuál es la tarea siguiente a T2; el servidor se encarga de recorrer el árbol que ya tiene para responderla.

La descripción programática de actividades en el Esquema de la Pizarra se ha llevado a cabo a través de la sintaxis descrita en [10] y utilizada en [11]. Esta descripción consiste en un modelo compuesto de los elementos Action, Routine, ask y Fragment. Estos elementos se describen en los subsiguientes apartados y cuyo código completo se puede leer en el anexo A.

Action

```
class Action extends RootClass{
      capability hasToBeRepeated ;
      capability isMultipleChoiceAction ;
}
```

Tabla 4.1: Esquema de Action

Se trata de una clase genérica o abstracta para englobar a todas las demás. La descripción de esquemas de la Pizarra permite esta particularidad, así como mecanismo de herencia simple, por la que una clase hija hereda todas las propiedades, capacidades y posibles relaciones de la clase padre. Con este propósito se ha modelado la clase "Action". Una clase abstracta de la que heredan Routine y Task.

Posee dos capacidades de carácter opcional:

- **hasToBeRepeated** Permite que una actividad pueda ser repetitiva
- **isMultipleChoiceAction** Permite que una actividad sea de selección múltiple

El propósito y funcionamiento de ambas capacidades es explicado en detalle en la sección 4.7

Routine

Representan las secuencias de pasos que los usuarios deben seguir para completar una actividad. Pueden ser representadas en QR Codes para que los usuarios los escaneen y se les muestren los pasos que las componen.

Como se ha dicho, una actividad (Routine) puede estar compuesta de uno o más pasos (Task) o de otras Routine. Para ello se establecen una serie de relaciones:

- **hasAction**: Representa una unión padre–hijo entre una Routine y otra Routine o Task

- **composesRoutine**: Inversa a hasAction. Es una unión hijo–padre.

- **previousAction**, **nextAction**: Sirven para indicar la Action (es decir, una Routine o Task) anterior o siguiente, respectivamente

Task

Es un modelo de cada uno de los pasos a seguir para completar la actividad. Como cada paso puede contener información de diferente tipo (un título, párrafos, imágenes, etc.), se dividen en fragmentos. Esta división se establece mediante relaciones "hasFragment". De manera similar a las Routines, cada Task puede ir precedida o seguida de otros pasos (Task o Routine). Para ello, como en el caso de las Routines, se emplean las relaciones "previousAction" y "nextAction".

Fragment

```
class Fragment extends RootClass{
      must properties{
            fragmentType;
            fragmentContent;
      }
}
```

Tabla 4.2: Esquema de Routine

Un Fragment contiene una porción de información relacionada con un paso de la actividad. Tiene dos propiedades estrechamente relacionadas:

- **fragmentType**: Define la naturaleza de la información. Puede ser título, texto, imagen, vídeo, etc.

- **fragmentContent**: Es el contenido en sí. En el caso de información textual, serán las palabras. En el caso de contenido multimedia (imágenes o vídeos) será un enlace (total o parcial) a la dirección de Internet donde se encuentra.

4.5. El cliente

Como se ha comentado al inicio de este capítulo, el cliente de este sistema es una aplicación para dispositivo móvil: en concreto, para Android. Se ha elegido implementar la aplicación para este sistema operativo debido a su alta penetración en la sociedad y el abanico de posibilidades que ofrece en cuanto al diseño de aplicaciones. Además, los dispositivos que tienen este sistema operativo cuentan con el hardware necesario para la aplicación que deseamos diseñar: cámara para escanear los códigos QR, memoria RAM para almacenar los datos de la aplicación, etc. Por otra parte, la parte servidor del sistema se ha implementado en lenguaje Java, y el lenguaje Android ofrece las máximas facilidades para integrar estas librerías en sus aplicaciones (al fin y al cabo, Android es lenguaje Java fundamentalmente).

Sin embargo, no tienen la misma capacidad de cómputo ni recursos que un equipo no móvil, así como otras restricciones (pantallas más pequeñas y duración de la batería) que habrá que considerar en este desarrollo. Existen dos puntos técnicos muy importantes a tener en cuenta en la implementación del cliente móvil:

Versionado: Android va ofreciendo nuevas versiones versiones de su sistema operativo periódicamente, que consisten en mejoras de rendimiento y organización del sistema, así como la reparación de errores y nuevas APIs para utilizar el potencial que los dispositivos van adquiriendo. Para el programador, esto se traduce en cambios de API de una actualización a otra. Estos cambios, que consisten en agregar nuevas funciones y dejar otras como obsoletas, suponen un reto para el programador, más aún cuando se desea abarcar cuantos más dispositivos mejor. Tal es nuestro caso, ya que los usuarios que utilizaran nuestra aplicación, lo harían desde su propio móvil o desde el que le fuese proporcionado por su educador, y la versión de Android del mismo podría ser cualquiera.

Variedad creciente de dispositivos: Continuamente salen al mercado nuevos dispositivos con distinto hardware. Aunque las nuevas APIs que acompañan estos cambios solucionen el acceso a esas nuevos elementos del dispositivos, la variedad de pantallas es un problema que corre a cuenta del programador. Más aún, teniendo en cuenta el auge de las *tablets*, que también ejecutan el sistema operativo Android y sus aplicaciones. En el diseño de esta aplicación se considerarán únicamente tamaños y resoluciones de pantalla pertenecientes a dispositivos móviles normales, intentando salvar en la medida de lo posible las variaciones (no tan acuciadas) de pantallas que existen en el mercado.

Para los test realizados en el AmILab, se han utilizado los dispositivos que se indican en la tabla 4.3

Modelo	HTC Desire	LG Optimus L5
Foto		
Versión del S.O.	2.3 (Gingerbread)	4.0 (Ice-Cream Sandwich)
Pantalla	3.7 pulgadas Multitáctil	4 pulgadas Multitáctil
Comunicaciones	GSM,3G,HSDPA WiFi b/g,Bluetooth	GSM/3G/HSDPA WiFi b/g,Bluetooth
Cámara	5 MPx, autofocus	5 MPx, autofocus

Tabla 4.3: Especificación técnica de los terminales empleados.

Diseñando la interfaz a medida que se observan los resultados en ambos terminales, cubrimos un especto aceptable de posibles dispositivoc móviles que los usuarios podrían tener (el rango de versiones entre uno y otro es bastante amplio, así como la diferencia entre las dimensiones de las respectivas pantallas).

4.5.1. Estructura del cliente

Esta parte cliente de AssisT-Task se compone de los siguientes elementos, esquematizados en la figura 4.4:

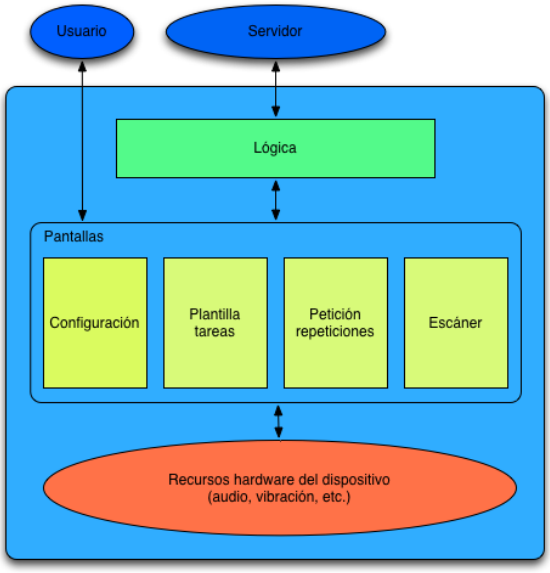


Figura 4.4: Esquema de la arquitectura del cliente

1. Pantallas o interfaces que se mostrarán al usuario:

   - **Pantalla de configuración**: permitirá establecer los ajustes que se explican en el apartado 4.6 para adaptar la aplicación a las necesidades y el contexto del usuario haciendo uso de los recursos multimodales del dispositivo. Estos ajustes son preferencias que se almacenan en el dispositivo y se comprueban cuando se van generando las tareas y se muestran al usuario.
   - **Pantalla de plantilla de tareas**: cada vez que se quiera mostrar un paso al usuario, se utilizará esta pantalla, cargada con los datos de los Fragments correspondientes a la tarea que se deben mostrar. Básicamente, consta de un campo textual y otro de imagen, además de los botones. Cuando se pulsa "siguiente" en una tarea, esta pantalla se limpia. El dispositivo realiza la petición al servidor de la siguiente tarea, y una vez obtenido dicha referencia, se piden sus Fragments. El contenido del Fragment textual se utiliza para rellenar el campo textual, y el contenido del Fragment de imagen es una URL para descargar la imagen del servidor, que se pintará en el campo de imagen de la interfaz. Esta descarga se realiza en segundo plano (Android ofrece facilidades para realizar la gestión de hilos), de forma que la descarga de esta imagen no bloquee la aplicación ni la detenga.
   - **Pantalla de introducción de repeticiones**: sirve para introducir el número de veces que se repetirá una tarea. Su propósito viene explicado en el apartado 4.7. Consta de un campo de texto con una cifra cuyo contenido se modifica con unos botones de incremento y decremento, para que no le suponga al usuario dificultad tener que usar repentinamente el teclado en la aplicación ni pueda introducir datos extraños.

- **Escáner**: es la pantalla que llama a la cámara del dispositivo para escanear los códigos QR que representan a una actividad. Hace uso de un decodificador externo de códigos QR ("Barcode Scanner"). La aplicación invoca la cámara del dispositivo para tomar la imagen del exterior a la vez que esta aplicación externa va comprobando en tiempo real si en la imagen hay algún código QR decodificable. Los códigos QR para esta aplicación contienen el nombre completo de una rutina. Con este dato, ya se puede pedir al servidor que construya una secuencia (a partir de los datos que pida a la Pizarra) cuyas tareas nos irá dando paso a paso.

2. La lógica de la aplicación. Debido al estilo de programación en Android, la lógica está repartida a lo largo de las diferentes pantallas que conforman la aplicación: la separación en el gráfico es una mera abstracción. Se encarga de realizar, según corresponda, las peticiones al servidor, así como de gestionar los registros que posteriormente serán almacenados y establecer los criterios de transición de una pantalla a otra. Por la cuestión del versionado que se comentaba al inicio de esta sección, esta lógica también hace la gestión de hilos, lo que evita hacer tareas de mucha carga computacional (peticiones a la Pizarra y descarga de imágenes) en el hilo gráfico principal. Éste es un requisito obligatorio para que la aplicación funcione en las últimas versiones de Android.

4.5.2. Interfaz de la aplicación

Los elementos de la interfaz de la aplicación han sido dispuestos partiendo de las consideraciones del trabajo previo [11] y las directrices de la FSDM. Por una parte, se ha hecho lo posible para que la interacción con el usuario sea la estrictamente necesaria: al iniciar la aplicación, simplemente tiene que escanear el código QR que corresponda a la actividad que desea hacer. A partir de ese punto, sólo tiene que ir navegando a lo largo de los pasos de la tarea, avanzando a medida que vaya completanto las distintas tareas. Sólo tendrá que intervenir de otra manera en dos casos: escoger una de las opciones de la selección múltiple o introducir un número de veces que se repetirá cierta tarea. Esto se explicará con detalle en la sección 4.7.

Visualmente, la interfaz deberá abarcar toda la pantalla, ocultando elementos como la barra de notificaciones o de acción, que podrían distraer al usuario.

En cuanto a la información en sí, se ofrece la posibilidad de ofrecerla de forma multimodal (imagen, audio, etc.) ya que es así como se definieron los Fragments, en forma de descriptor de contenido y contenido en sí. También se ha implementado la opción de leer en voz alta las instrucciones al usuario.

En la imagen 4.5 se muestra el flujo de la aplicación. Cuando se abre la aplicación,

se muestra automáticamente la pantalla de captura de códigos QR (a través de la aplicación "Barcode Scanner" [3] y sus librerías "XZing Library" [4]). El usuario simplemente tiene que apuntar con el móvil hacia donde esté el código. La aplicación pide al servidor la primera tarea de la rutina perteneciente al código que tiene que escanear (el servidor, por su parte, actúa como describimos en la sección 4.4.2. La aplicación, al recibir esta información, genera la interfaz correspondiente. Para ello, pide al servidor los Fragments por los que está compuesta la tarea y genera los elementos gráficos pertinentes según el tipo de Fragment (texto, imagen, etc.).

Un ejemplo de interfaz de la aplicación es el que se muestra en la imagen 4.6. Como se puede ver, se han ocultado las barras superiores del sistema operativo, quedando la interfaz completamente limpia de posibles distracciones. Estas interfaces de ejemplo pertenecen a un ejemplo cuyo código completo puede leerse en el anexo B. Los elementos que hay son los siguientes:

- La acción a realizar es un Fragment de tipo Text perteneciente a esta tarea. La aplicación puede configurarse para que se lea este texto al cargar la tarea, al tocar la pantalla, o tras un tiempo de inactividad del usuario.
- La imagen representativa sirve de muestra y apoyo al usuario para completar la tarea.
- En la parte inferior de la pantalla se muestran los botones para navegar hacie al paso anterior o siguiente. Si se trata de la primera tarea, sólo es visible (y de mayor tamaño) el botón de siguiente tarea. En la última tarea, el botón de siguiente se cambia por uno que finaliza la aplicación (la imagen que acompaña al botón da idea de finalización correcta de la tarea, lo que supone un refuerzo positivo). A medida que el usuario va pulsando los botones de siguiente, la aplicación pide al servidor la tarea cuya interfaz construirá. Dado el usuario influye en la secuencia de tareas que se van a realizar, es más coherente que le móvil almacene un historial de referencias a tareas que hayan sido ejecutadas, y por tanto, pedir la tarea anterior consiste únicamente en navegar en este historial.

Al finalizar la aplicación, se cierran las conexiones y se guarda el registro de ejecución.

Registro de ejecución

Tener informados a los educadores de la actividad de estos usuarios es fundamental por varios motivos. Por una parte, es una información muy valiosa de cara a construir

[3] Disponible de forma gratuita en el Android Market
[4] http://code.google.com/p/zxing

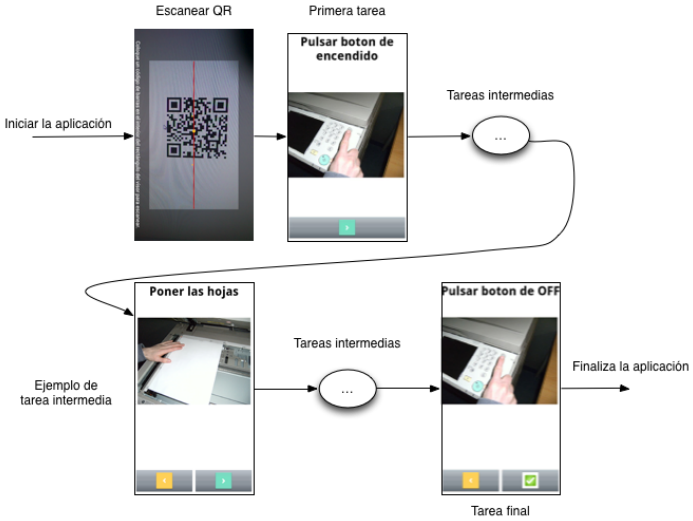


Figura 4.5: Flujo de la aplicación

Figura 4.6: Ejemplo de interfaz

nuevas tareas para estos usuarios, sabiendo qué les ha resultado difícil o en qué tareas se desenvuelven mejor. Esto último también sirve como información del progreso del usuario.

Es por esto que la aplicación guarda unos registros de los tiempos y tareas que ha realizado el usuario a través de la aplicación, así como de las alarmas que haya hecho saltar el usuario por estar inactivo demasiado tiempo y el tipo de las mismas. Este registro es enviado al servidor y almacenado en la propia terminal, además está creado

con una sintaxis fácilmente legible por ordenador, en caso de querer implementar un sistema que lo hiciese.

4.6. Adaptación al usuario

La aplicación está diseñada para tener en cuenta el contexto y las necesidades del usuario. A nivel de aplicación, esto se traduce a las siguientes opciones de configuración accesibles desde el menú de contexto de la aplicación, como se muestra en la figura 4.7:

Nombre: se puede introducir este dato en la aplicación, de forma que aparecerá en los registros de salida, facilitando su organización y análisis

Time Out: se trata de la duración máxima de una tarea. Si el usuario permanece inactivo en una misma actividad, se produce un aviso.

Tipo de alarma: se puede configurar el tipo de aviso que se producirá cuando expire el tiempo máximo de inactividad en una tarea (vibración, sonido o lectura del texto de la aplicación).

Lectura de las tareas: esta opción se escogerá si se desea que el generador de voz del dispositivo lea la descripción de las tareas cuando las cargue. Esta opción permite que usuarios con problemas visuales o de lectura también puedan utilizar la aplicación.

Dirección IP y puerto del servidor: si una organización determinada monta y utiliza su propio servidor de AssisT-Task, pueden configurar la aplicación para ser utilizada ahí. Por defecto, la aplicación se sirve del servidor de AssisT-Task que se ejecuta en el AmILab.

Por otra parte, existe un mecanismo que permite que la tarea siguiente a otra difiera según una propiedad de dicha relación cuyo valor es el nombre de usuario al que se adapta ese paso. En la tabla 4.4 se muestra un ejemplo de tarea programada con desvíos si el usuario que está utilizando la aplicación es *Juan*.

4.7. Control de flujo de la secuencia de tareas

Generar interfaces ubicuas para un conjunto de tareas cuya secuencia de pasos es necesariamente lineal supone un apoyo insuficiente. Normalmente, cualquier tarea de

```
entity Task:tEjemploAdaptacion@amilab{
        relations{
                composesRoutine = Routine:rEjemplo@amilab;
                prevAction = Task:TSiguiente@amilab;
                nextAction = Task:tAdaptada@amilab{ user = ''Juan'' };
        }
}
```

Tabla 4.4: Ejemplo de tarea con desvío para un usuario

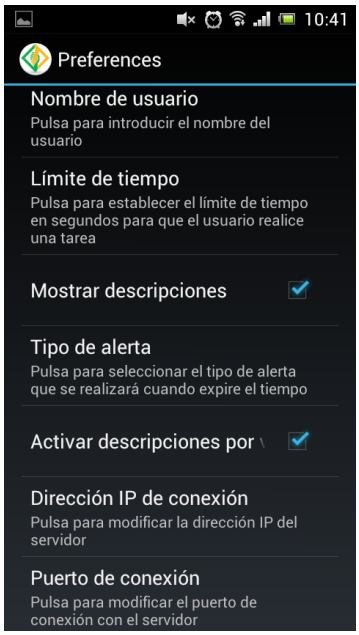


Figura 4.7: Pantalla de configuración de la aplicación

la vida cotidiana comprende tareas que varían dependiendo de detalles concretos sobre lo que se desea hacer (por ejemplo, el uso de una fotocopiadora varía dependiendo de si el usuario desea hacer copias a una o dos caras). Debido a esto, se ha introducido en el sistema la opción de introducir tareas repetitivas y de selección múltiple.

4.7.1. Tareas repetitivas

La necesidad de introducir tareas repetitivas radica en la ventaja que supone el guiado de tipo mecánico para este tipo de personas. No hay que olvidar que se trata de una herramienta de aprendizaje, por lo que es de interés que las actividades que comprendan tareas que hay que repetir varias veces consecutivas durante la secuencia sean descritas al paciente como tales, en lugar de ofrecer una única tarea que dé esta información de forma descriptiva (esto es,"pulsa cierto botón" cuatro veces de forma

consecutiva, en lugar de "pulsa el botón cuatro veces", y pasar a la siguiente tarea). La idea es que las tareas propuestas sean de carácter atómico, con el objetivo de poder conformar con ellas actividades de mayor complejidad, que el usuario aprenderá a realizar por el uso y aprendizaje, en lugar de por comprensión global inicial de la actividad, como se intentaría hacer en el caso de una persona sin discapacidad. Sin embargo, el usuario puede sentir incertidumbre si observa que un paso concreto se repite durante la secuencia (puede llegar a sospechar que el funcionamiento es erróneo) para un número de repeticiones algo más elevado, por lo que es un requisito indispensable que se muestre en la interfaz un indicador de progreso. En las interfaces generadas por AssisT-Task, este progreso es mostrado en forma del número de repeticiones que quedan por realizar.

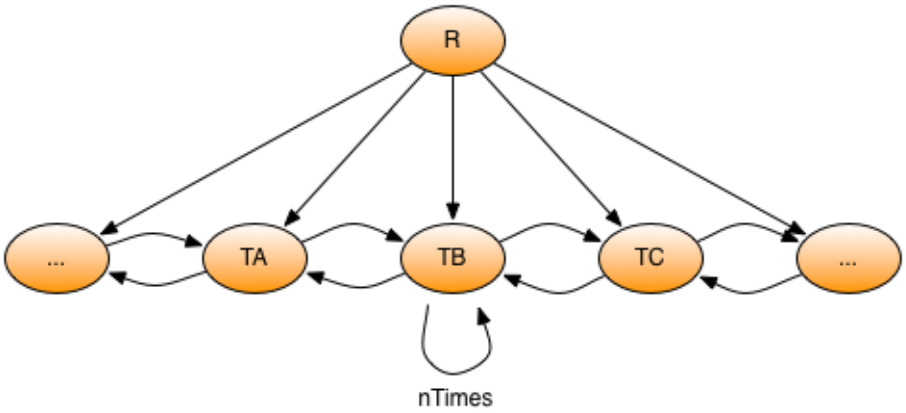


Figura 4.8: Árbol de rutinas para el ejemplo de tarea repetitiva

El número de repeticiones que puede ir asociado a una tarea es, generalmente, decidido por el tutor cuando planea la actividad. Sin embargo, también se proporciona funcionalidad para que el tutor decida durante esa planificación que se deje esa decisión en manos del usuario. Esto significa, que al llegar al paso en cuestión, la aplicación móvil preguntará al usuario cuántas veces desea repetir la tarea que acaba de realizar. De esta manera se cubre la necesidad de ofrecer, además, un modo de aprendizaje más avanzado, que incluye al usuario durante el proceso de realización de una tarea, en lugar de proporcionarle un guiado más absoluto y pasivo.

Una tarea repetitiva se describe como se muestra en la tabla 4.5.

Este ejemplo describe con el lenguaje de la Pizarra la tarea TB del árbol de rutinas de la figura 4.8. La Tarea TB es una tarea que se repetirá cinco veces hasta pasar a la tarea especificada como tarea siguiente (Tarea TC).

Como se comentaba en la sección 4.4.2, esta funcionalidad se traduce en una capacidad **hasToBeRepeated**, con una propiedad **nTimes** de tipo numérico, en la que se introduce el número de repeticiones que se desean hacer. Si se desea que las elija

```
entity Task:TB@amilab{
        capabilities{
                hasToBeRepeated{
                        nTimes=5;
                }
        }
        relations{
                composesRoutine = Routine:R@amilab;
                hasFragment = Fragment:fPrueba1@amilab;
                hasFragment = Fragment:fPrueba2@amilab;
                prevAction = Task:TA@amilab;
                nextAction = Task:TC@amilab;
        }
}
```

Tabla 4.5: Ejemplo de tarea repetitiva

el usuario, esta propiedad deberá tener el valor *0*. En el móvil, cuando se llegue a esa tarea, se preguntará al usuario cuántas más veces quiere repetirla, en la interfaz mostrada en la figura 4.9. Tanto si las repeticiones las marca el usuario como si vienen ya definidas en la entidad, este dato se almacena en el servidor, en forma de contador que se va decrementando a medida que el cliente va pidiendo las tareas siguientes, las cuales serán la misma hasta que dicho contador llegue a cero. En el diagrama mostrado en la figura 4.10 podemos ver la comparativa entre cómo ocurre una petición de tarea cuando ésta es lineal (caso *a*) y cuando es repetitiva (ésta, a su vez, puede tener las repeticiones fijadas de inicio, como en el caso *b* o dejarle esa decisión al usuario, como en el caso *c*).

Se puede observar que los colores de esta pantalla difieren mucho de los del resto de la aplicación. Esto puede llamar la atención al usuario, pero en este caso es bastante conveniente, porque en esta pantalla el usuario tiene que introducir un dato, y esta llamada de atención probablemente le haga fijarse más en el dispositivo y darse cuenta de que debe interactuar. Además, esta interfaz está programada para que el campo de texto donde aparece el número de repeticiones no sea directamente editable, evitando que el usuario pueda introducir cualquier cosa o confundirse con la aparición del teclado. Los botones de incremento y decremento de la cifra son los que deberá usar el usuario para especificar las repeticiones.

En la figura 4.11 se observa una tarea que se está repitiendo. Se puede apreciar la aclaración que se hace al usuario acerca del progreso, para evitar que perciba incertidumbre o crea que la aplicación ha repetido una tarea arbitrariamente y no funciona.

Figura 4.9: Ejemplo de interfaz para introducir el número de repeticiones

4.7.2. Bifurcaciones

Muchas actividades que realizamos en la vida cotidiana son objeto de decisiones que se van tomando durante la realización de las mismas (por ejemplo, siendo una actividad hacer la colada, la secuencia de pasos varía dependiendo de si se decidiese lavar ropa blanca, lavar ropa de color o lavar ropa delicada).

Sin el uso de bifurcaciones, esas variaciones que se producen en la secuencia de tareas implicarían planificar tareas diferentes para englobar todas las posibles variantes (si queremos modelar entidades para que el usuario haga la colada, se trataría de hacer una actividad para hacer la colada para ropa de color, hacer la colada para ropa delicada o hacer la colada para ropa blanca). En principio, no parece una mala solución al problema. Sin embargo, no olvidemos que estamos tratando un caso particular en el que la secuencia se vería alterada a partir de un único punto de decisión. Sin embargo, esto no tiene por qué ser así: ¿qué pasaría si existen varios puntos de este tipo? Resultaría inviable replicar tantas actividades como posibles bifurcaciones (y las combinaciones entre ellas).

En AssisT-Task, el tutor puede planificar para el usuario tareas de selección múltiple, en las que habrá pasos de la secuencia en los que se muestran al usuario las posibles opciones a escoger, variando las tareas sucesivas en función de la opción escogida (en el ejemplo que comentábamos anteriormente, la actividad global sería hacer la colada, y consistiría en una serie de pasos, entre ellos uno del tipo escoger el modo de lavado, en

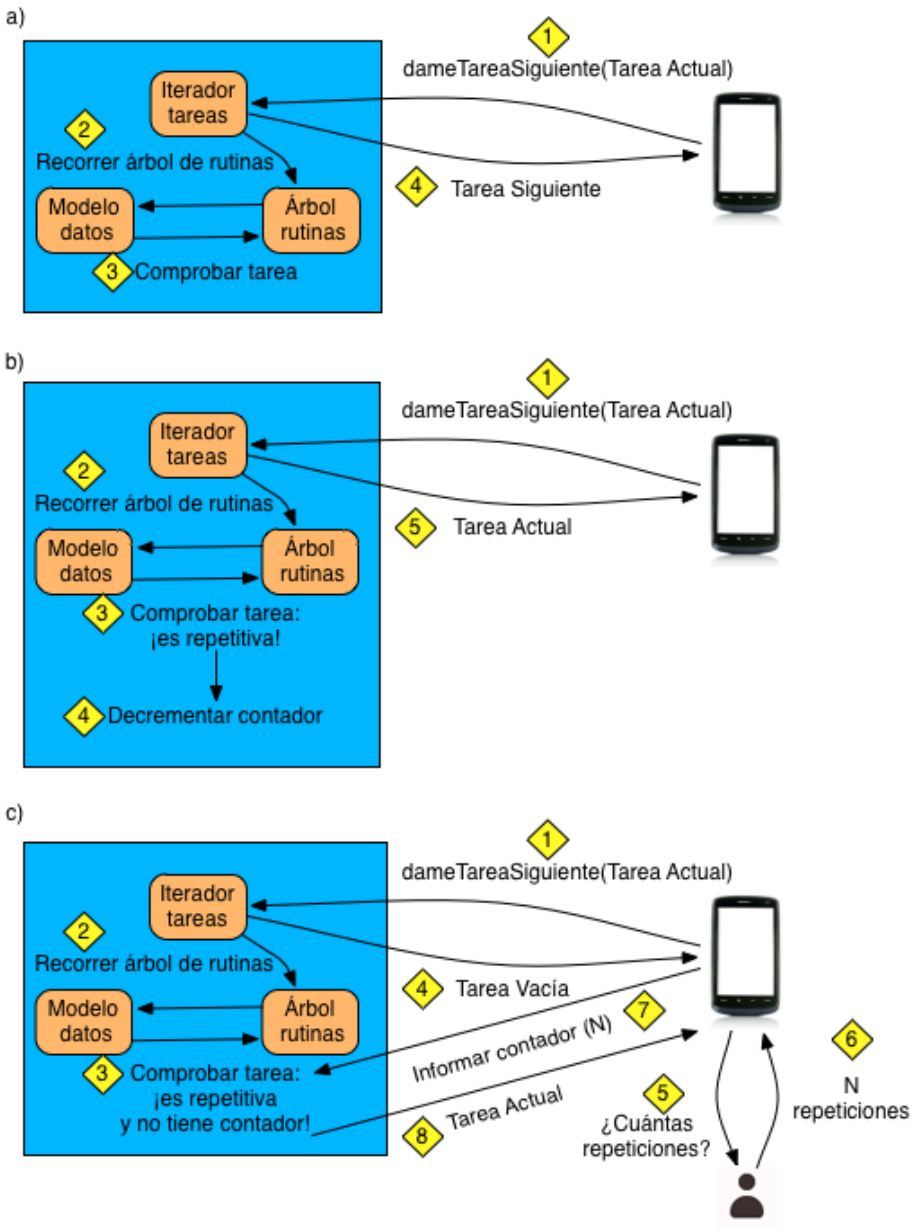


Figura 4.10: Interacción cliente-servidor para tareas lineales (a) y repetitivas (b y c)

el que se mostrarían las opciones ropa blanca, ropa de color y ropa delicada, variando la sub-secuencia sucesiva según la opción).

Tenemos en la figura 4.12 un caso de actividad en la que se realiza una selección múltiple. Una vez la secuencia ha llegado a la tarea TA (cuyo código se muestra en la tabla 4.6), vemos que existen varias relaciones de tarea siguiente, acompañadas de un campo **forOption** que indica a qué opción se corresponde. Es decir, el servidor, cuando llega esta tarea de selección, la envía al móvil (como se hace normalmente).

Figura 4.11: Ejemplo de interfaz con una tarea repetitiva en curso

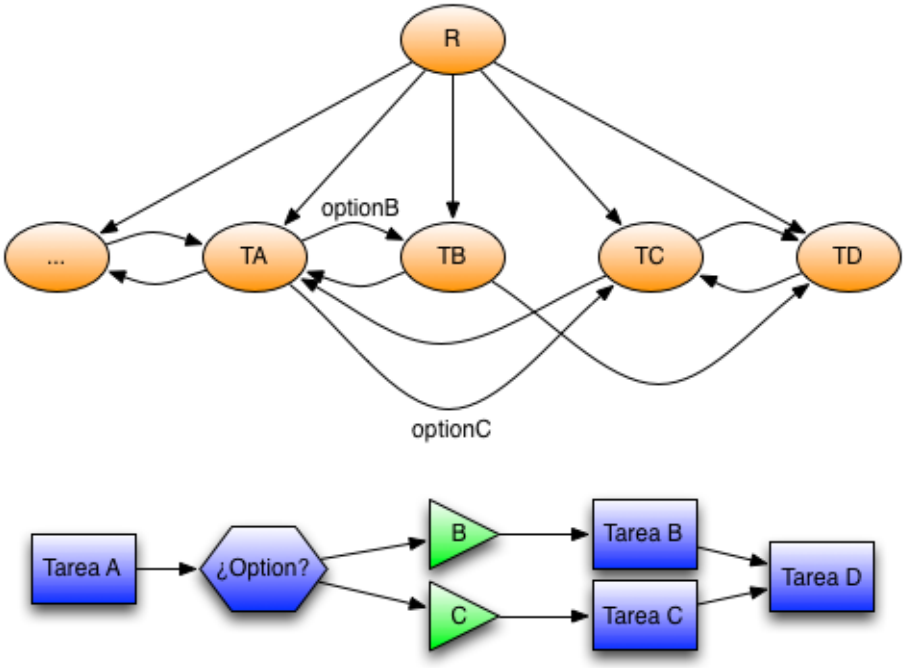


Figura 4.12: Árbol de rutinas para el ejemplo de tarea con selección múltiple

Éste, comprobará si la tarea tiene opciones , y se las pedirá al servidor, quien se las enviará en forma de lista de opciones (es decir, los valores de la propiedad multivaluada **options** de su capacidad **isMultipleChoiceAction**). Estas opciones se despliegan

ante el usuario como se muestra en la figura 4.13. A continuación, cuando se pulse el botón de tarea siguiente, la opción escogida es enviada al servidor junto con la petición, y éste utiliza dicha información para saber qué tarea del árbol ofrecer a continuación (aquella referenciada por la relación **nextAction** que contenga la propiedad **forOption** con el valor que se haya recibido del usuario).

Este proceso viene reflejado en el diagrama de la figura 4.14.

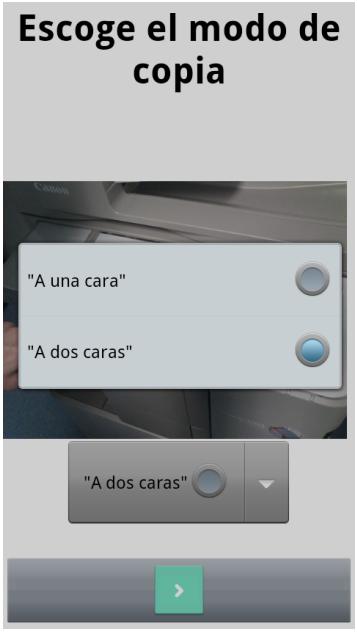


Figura 4.13: Ejemplo de interfaz de una tarea de selección múltiple

```
entity Task:TA@amilab{
	capabilities{
		isMultipleChoiceAction{
			options = (''Option B'', ''Option C'');
		}
	}
	relations{
		composesRoutine = Routine:R@amilab;
		hasFragment = Fragment:fPrueba1@amilab;
		hasFragment = Fragment:fPrueba2@amilab;
		prevAction = Task:TaskNull@amilab;
		nextAction = Task:tB@amilab { forOption = ''Option B''}
		nextAction = Task:tC@amilab { forOption = ''Option C''}
	}
}
```

Tabla 4.6: Ejemplo de tarea de selección múltiple

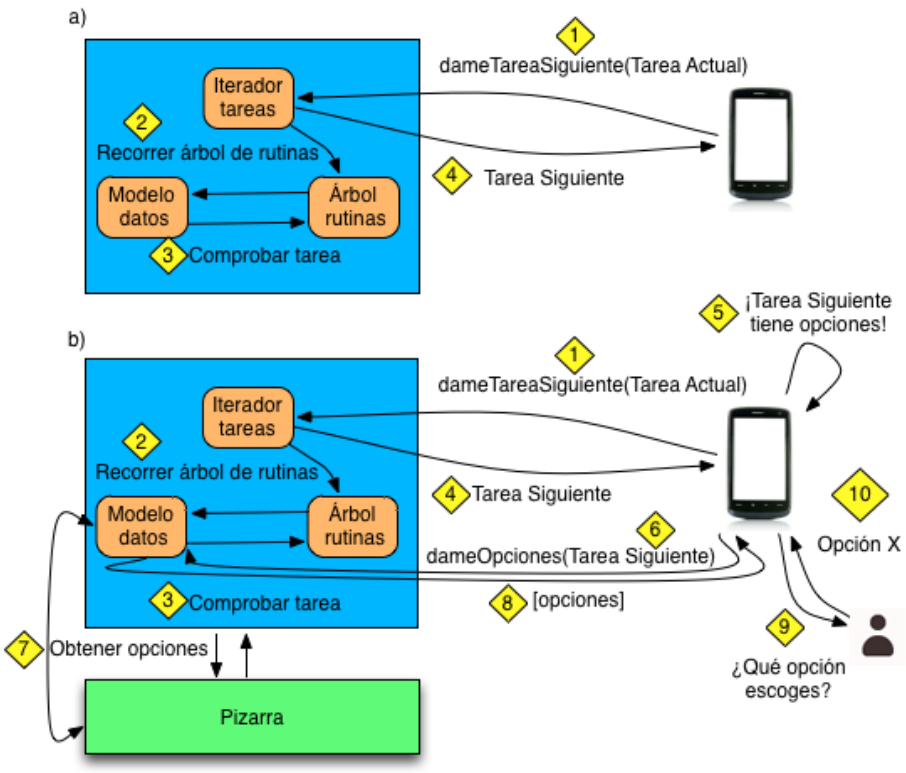


Figura 4.14: Interacción cliente-servidor para tareas lineales (a) y de selección múltiple (b)

5 Ejemplo de uso

5.1. Introducción

En este capítulo se presenta un ejemplo completo de una actividad que se ha implementado para el sistema AssisT-Task.

Se desea generar un manual paso a paso para hacer fotocopias. El manejo de la máquina fotocopiadora es complejo para un individuo con discapcidad intelectual. Por ello, queremos indicarle con AssisT-Task qué pasos tiene que seguir para realizarlas.

La actividad que tendría que realizar el usuario es la descrita en al figura 5.1.

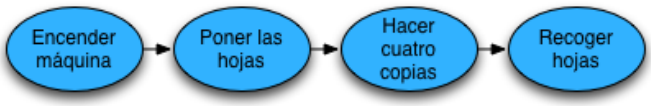


Figura 5.1: Planteamiento de actividad

Sin embargo, la actividad debe estar adaptada a usuarios con discapacidad cognitiva: las tareas deben ser de carácter atómico, y las descripciones sencillas. Esto se traduce en una mayor granularización de los pasos, que ya se justificó en el capítulo 4. La secuencia de tareas que se desea implementar, por tanto, quedaría descrita como se muestra en la figura **??**

Para que esta secuencia de pasos sea posible, en nuestro sistema tenemos que definir, con la sintaxis descrita en el apartado **??** un árbol con las siguientes características:

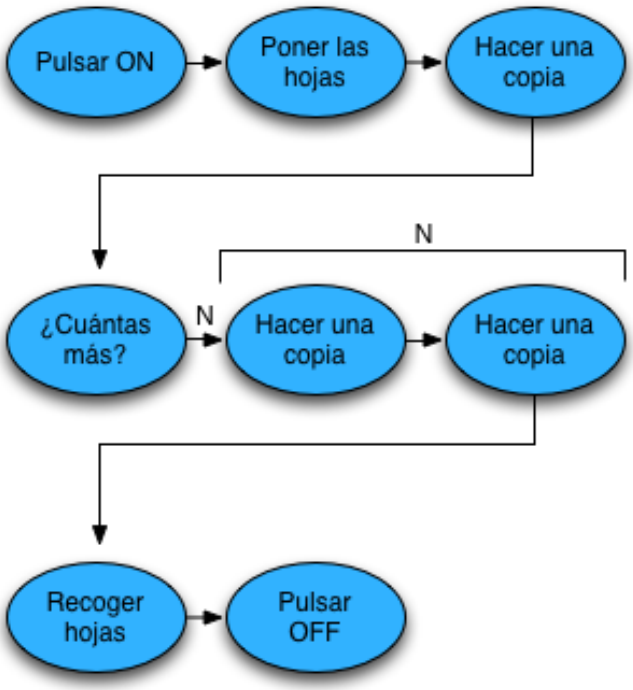


Figura 5.2: Planteamiento adaptado de actividad

- Una Routine que englobe todos los pasos de la secuencia en forma de hijos.
- Una Task hija por cada tarea de las descritas, excepto las de hacer copias, que serán una única tarea cuya propiedad **nTimes** de su capacidad **hasToBeRepeated** valdrá *0*. Esto indica que las repeticiones (copias, en nuestro caso) las decidirá el usuario.
- Cada una de esas Task, deberá tener Fragments con la información que se desee proporcionar con la tarea. En este ejemplo particular, vamos a mostrar una descripción textual seguida de una imagen.

Este árbol se puede ver en la figura 5.3. Cada una de las Task (en color verde) están relacionadas con una Task previa y una Task siguiente, al igual que la Routine (de color rojo) aunque ambas relaciones son nulas para nuestro ejemplo. Las Task, además están relacionadas con un Fragment textual con la descripción de la tarea y otro con la ruta a una imagen explicativa de la tarea (ambos, de color azul en la figura). Se podrían añadir más Fragments con más información sobre cada Task, pero se ha decidido elaborar las tareas con esos dos refuerzos descriptivos. Los Fragments también están enlazados por relaciones previo/siguiente.

Una vez codificada la secuencia de pasos que queremos generar (el código completo

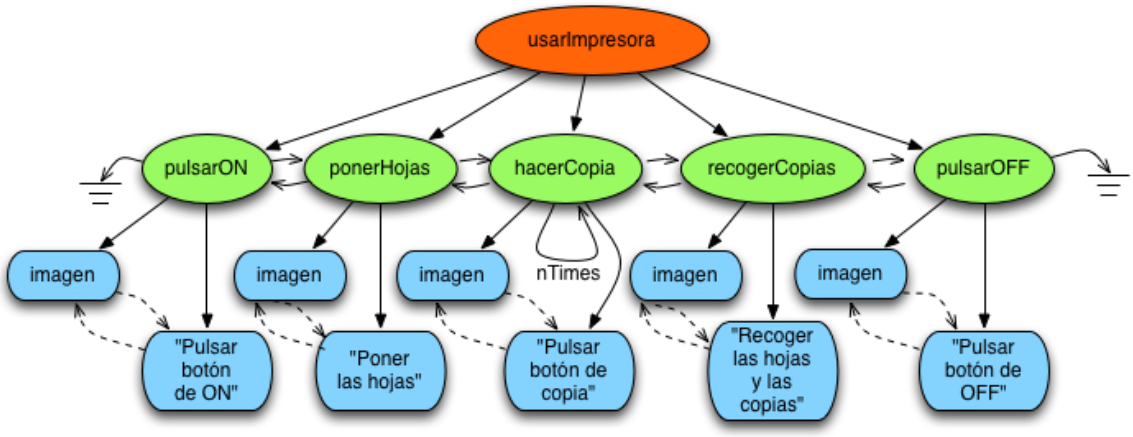


Figura 5.3: Árbol correspondiente a la actividad de las fotocopias

se encuentra en el Anexo B), tenemos que generar un código QR que contenga el nombre completo de la rutina que hemos creado: *Routine:usarImpresora@amilab*. Este código es el que tenemos que escanear con el móvil cuando iniciemos la aplicación. La secuencia de pantallas se inicia como se muestra en al figura 5.4.

Figura 5.4: Pasos iniciales del ejemplo

¿Cómo está actuando el sistema para ir generando esos pasos?

1. La aplicación ha leído el código QR correspondiente a **usarImpresora**.
2. Se notifica al servidor y éste construye el árbol que descibimos anteriormente
3. El móvil pide la primera tarea. El servidor recorre el árbol en profundidad y encuentra la primera tarea: **pulsarON**. Se envía al móvil.
4. El móvil pide la tarea siguiente a **pulsarON**. El servidor comprueba que la tarea siguiente a la actual es **ponerHojas**, y la envía.

5. Mismo procedimiento que en el caso anterior para pasar a **hacerCopia**

La útima tarea que se ha realizado es repetitiva, y con decisión del usuario, es decir, que ahora la aplicación deberá solicitar al usuario el número de veces que debe repetirse la tarea. La secuencia de pasos que se producen continúa como se muestra en la figura 5.5.

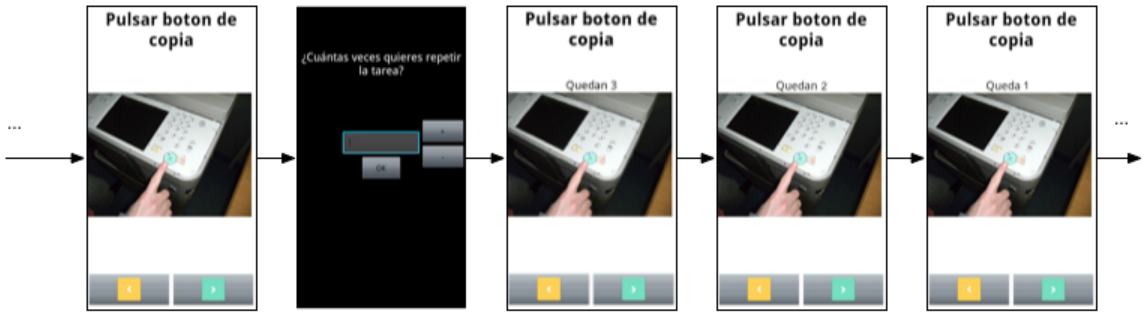


Figura 5.5: Pasos intermedios del ejemplo

El sistema ha actuado de la siguiente manera:

- El dispositivo nos había pedido la siguiente tarea a **hacerCopia**. Sin embargo, se trataba de una tarea repetitiva con **nTimes** a valor *0*. Cuando el servidor ve esto, en lugar de enviarle la siguiente tarea al móvil, le envía una tarea vacía.
- El móvil interpreta esta tarea vacía como una señal para preguntarle al usuario cuántas veces más quiere repetir la tarea que acaba de realizar. Se despliega la pantalla de introducción de repeticiones, y se envía este dato al servidor.
- El servidor almacena este dato como contador, y va devolviendo la tarea que se había realizado anteriormente cada vez que el móvil le pide una siguiente hasta qeu el contador se agota.
- Una vez el contador llega a cero, el servidor devuelve al dispositivo la siguiente tarea a la que se estaba repitiendo, según dicte el árbol.

A partir de aquí, se procede hasta el final de la misma manera (son todas tareas lineales), como se muestra en la figura 5.6

En la tarea **pulsarOFF**, el servidor ya no encuentra tareas siguientes. El móvil entiende esto como el fin de la secuencia, y la aplicación se cierra.

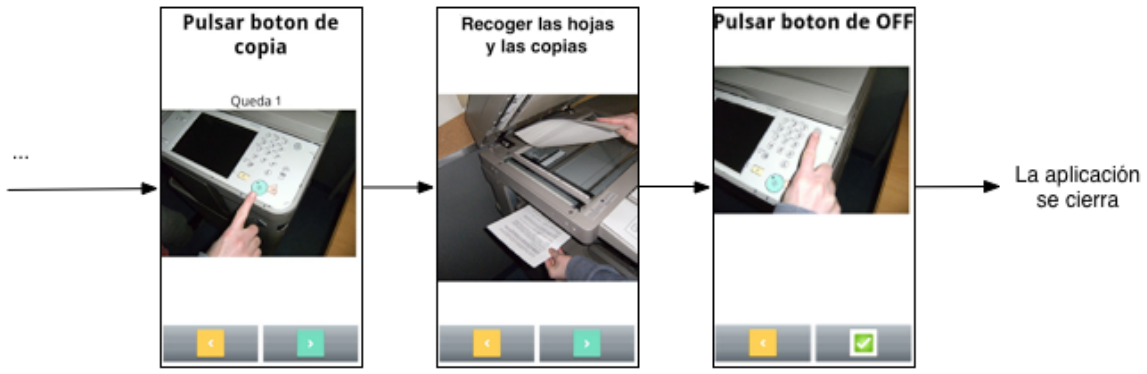


Figura 5.6: Pasos finales del ejemplo

6 Conclusiones y Trabajo Futuro

6.1. Conclusiones

Este Trabajo de Fin de Grado parte de trabajos previos [11], y está diseñado para las personas con discapacidad intelectual en la realización de actividades de la vida diaria.

Este tipo de tareas comprenden un abanico muy amplio de destrezas de las que estas personas pueden carecer, y que supone que presenten dificultades para realizar tareas de la vida cotidiana. En muchas ocasiones es la ayuda de sus cuidadores la que permite que puedan realizarlas, o en el mejor de los casos, aprenderlas y poder recrearlas de forma autónoma. Sin embargo, esta labor de los educadores supone un gasto económico, logístico y temporal para unas personas que tienen que atender a un número amplio de personas. El sistema en el que se basa este trabajo pretende proporcionar soporte a personas con discapacidad cognitiva para que adquieran dichas destrezas sin la intervención directa y constante de un educador, pudiendo éstos asistir a más personas y de manera más ágil.

Este sistema utiliza dispositivos móviles y códigos QR para generar manuales interactivos que les guían paso a paso. La elección del teléfono móvil viene motivada por su alta penetración en la sociedad y la funcionalidad tan amplia que ofrecen los dispositivos actuales. Además, constan de *hardware* que permite que las instrucciones generadas puedan ser presentadas de forma multimodal (texto, vídeo, audio, etc.). De esta forma, las actividades implementadas de esta manera se adaptan a las necesidades del usuario.

Esta adaptación es la principal motivación de este trabajo: se estudian nuevas nece-

sidades del usuario, y se rediseña el sistema para que se adapte al contexto y necesidades del usuario. De esta forma, se ofrece funcionalidad para dos tipos nuevos de tarea: tareas con repeticiones y de selección múltiple. Estas nuevas tareas permiten que los manuales ya no son una secuencia lineal de pasos. Por una parte, se permite que el usuario participe durante el proceso: puede elegir cuántas veces repetir las tareas y escoger una opción de entre una lista. Estas elecciones influirán en los siguientes pasos que se le indicarán. Esta nueva funcionalidad, añadida a la que ya existía consistente en adaptar tareas por código a determinados usuarios, suman un abanico de posibilidades que permite una adaptación muy enriquecedora para el aprendizaje del usuario y su adquisición progresiva de autonomía.

Por otra parte, se permite una adaptación al usuario en forma de configuración de la aplicación, aprovechando las ventajas multimodales del dispositivo para especificar si las tareas se han de leer en voz alta generada por el móvil para los usuarios que tengan dificultades para leer el texto o activar alarmas con temporizador de distintos tipos por si el usuario se bloquea. De cara a la flexibilidad de la aplicación, se permite configurar las direcciones de red del servidor al que pedir las tareas, pudiendo montar varios servidores para distintos colectivos u organizaciones que utilizasen este sistema.

Se ha mantenido contacto con expertos de la Fundación de Síndrome de Down de Madrid para establecer pautas y directivas en cuanto a la disposición de los elementos de las interfaces gráficas que se le muestran al usuario.

Viendo el resultado final, la aplicación que se ha implementado cumple los objetivos marcados, y conforma un sistema que, efectivamente, puede utilizarse para ayudar a estas personas a adquirir autonomía progresivamente para realizar sus tareas de la vida diaria.

6.2. Trabajo Futuro

Estas son las líneas en las que se podría trabajar próximamente en relación con el trabajo desarrollado en este TFG:

- Actualmente se está trabajando en el AmILab en una herramienta de autor para generar el código perteneciente a las entidades que compnen las tareas, así, como añadir en tiempo real esas actividades a la Pizarra, siendo desde ese momento accesibles para ser utilizadas en cualquier móvil que tenga instalado AssisT-Task. Esta herramienta servirá a los educadores para organizar y planear las actividades de los usuarios a los que asisten a través de una interfaz gráfica intuitiva y sencilla.
- Los expertos de la FSDM sugirieron hacer una versión de la aplicación móvil de

AssisT-Task para tablets, unos dispositivos que, aseguran, son bastante intuitivos y fáciles de usar y manejar para usuarios con discapacidad cognitiva.

- Como se comentaba en el capítulo 3, algunos trabajos actuales utilizan motores de análisis de logs basados en inteligencia artificial y otras heurísticas para extraer datos de los registros de salida de la aplicación. Elaborar un sistema de este tipo sería interesante de cara al futuro.

- Sería interesante poder extraer conclusiones de una evaluación del sistema con un grupo de usuarios con discapacidad intelectual.

- También se están llevando a cabo en el laboratorio otros proyectos relacionados con la asistencia de individuos con dicapacidad intelectual en distintos aspectos de su vida. En concreto, ya ha sido desarrollado un sistema de guiado en interiores (basado también en códigos QR) y otro de navegación en exteriores (basado en navegación GPS). Un trabajo muy interesante sería englobar los tres trabajos en un sistema de gestión integral de la vida diaria de un usuario con estas características.

Bibliografía

[1] AdastraSoft. iPACS. online at http://www.adastrasoft.com/, 2009.

[2] D. Braddock, M.C. Rizzolo, M. Thompson, and R. Bell. Emerging technologies and cognitive disability. *Journal of Special Education Technology*, 19(4):49–56, 2004.

[3] S. Carmien. Task support for people with cognitive impairments and their caregivers. *American Journal of Occupational Therapy*, 14(3):1–4, 2004.

[4] Stefan Carmien. End user programming and context responsiveness in handheld prompting systems for persons with cognitive disabilities and caregivers. In *CHI'05 extended abstracts on Human factors in computing systems*, pages 1252–1255. ACM, 2005.

[5] Stefan Carmien, Rogério DePaula, Andrew Gorman, and Anja Kintsch. Increasing workplace independence for people with cognitive disabilities by leveraging distributed cognition among caregivers and clients. *Computer Supported Cooperative Work (CSCW)*, 13(5-6):443–470, 2004.

[6] Y.J. Chang, W.C. Chang, and T.Y. Wang. Context-aware prompting to transition autonomously through vocational tasks for individuals with cognitive impairments. In *Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility*, pages 19–26. ACM, 2009.

[7] D.K. Davies, S.E. Stock, and M.L. Wehmeyer. Enhancing independent task performance for individuals with mental retardation through use of a handheld self-directed visual and audio prompting system. *Education and Training in Mental Retardation and Developmental Disabilities*, 37(2):209–218, 2002.

[8] R. Engelmore and T. Morgan. *Blackboard systems*. Addison-Wesley Reading, MA, 1988.

[9] F. Furniss, G. Lancioni, N. Rocha, B. Cunha, P. Seedhouse, P. Morato, and M.F. O'Reilly. Vicaid: Development and evaluation of a palmtop-based job aid for workers with severe developmental disabilities. *British Journal of Educational Technology*, 32(3):277–287, 2001.

[10] J. Gomez, G. Montoro, P.A. Haya, M. García-Herranz, and X. Alamán. *Ubiquitous Developments in Ambient Computing and Intelligence: Human-Centered Applications,*, chapter 20, pages 205–218. IGI-Global, 2011.

[11] Javier Gómez, Germán Montoro, Pablo A Haya, Xavier Alamán, Susana Alves, and Mónica Martínez. Adaptive manuals as assistive technology to support and train people with acquired brain injury in their daily life activities. *Personal and Ubiquitous Computing*, pages 1–10, 2012.

[12] Handhold Adaptive. iPrompts. Visto online en http://www.handholdadaptive.com/iprompts.html (Septiembre 2011), 2009.

[13] Eva Hidalgo, Luis Castillo, R Ignacio Madrid, Óscar García-Pérez, MR Cabello, and J Fdez-Olivares. Athena: Smart process management for daily activity planning for cognitive impairment. In *Ambient Assisted Living*, pages 65–72. Springer, 2011.

[14] International Standards Organization. *ISO 13407. Human Centred Design Process for Interactive Systems*. Geneva, Swiss, 1999.

[15] Richard Levinson. The planning and execution assistant and trainer (peat). *The Journal of head trauma rehabilitation*, 12(2):85–91, 1997.

[16] E.F. LoPresti, A. Mihailidis, and N. Kirsch. Assistive technology for cognitive rehabilitation: State of the art. *Neuropsychological Rehabilitation*, 14(1):5–39, 2004.

[17] Linda C Mechling. Assistive technology as a self-management tool for prompting students with intellectual disabilities to initiate and complete daily tasks: A literature review. *Education and Training in Developmental Disabilities*, 42(3):252, 2007.

[18] UNE Norma. en iso 9999: 2007. *Productos de apoyo para personas con discapacidad. Clasificación y terminología*.

[19] Katherine M. Tsui and Holly A. Yanco. Prompting devices: A survey of memory aids for task sequencing. In *QoLT International Symposium: Intelligent Systems for Better Living, held in conjunction with RESNA 2010*, 2010.

[20] David Wechsler. *WAIS-III: Wechsler adult intelligence scale*. Psychological Corporation San Antonio, 1997.

A Modelado de las actividades en el Esquema

```
# Action
class Action extends RootClass{
      capability hasToBeRepeated;
      capability isMultipleChoiceAction;
}
# Routine
class Routine extends Action;
# Task
class Task extends Action;
# Fragment
class Fragment extends RootClass{
      must properties{
            fragmentType;
            fragmentContent;
      }
}

capability hasToBeRepeated extends RootCap{
      must properties{
            nTimes;
      }
}

capability isMultipleChoiceAction extends RootCap{
      must properties{
            options;
```

```
        }
}

enum FragmentType = ("title", "description", "video", "audio", "image");
property fragmentType String = "title";
property fragmentContent String;
property userName String = "default";
property nTimes Numeric;
property options String = "default";
property forOption String;

relation hasAction extends RootRel{
        range Routine;
        domain Action;
        card 1:*;
        #inverse_of composesRoutine;
}


relation composesRoutine extends RootRel{
        range Action;
        domain Routine;
        card 1:*;

        #inverse_of hasTask;
}

relation hasFragment extends RootRel{
        range Task;
        domain Fragment;
        card 1:*;

        #inverse_of composesTask;
}

relation composesTask extends RootRel{
        range Fragment;
        domain Task;
        card 1:*;

#       inverse_of hasFragment;
}

relation nextAction{
```

```
        range Action;
        domain Action;
        card 1:*;
        must properties{
              userName = "default";
        }
        may properties{
              forOption;
        }
}
relation prevAction{
        range Action;
        domain Action;
        card 1:*;
        must properties{
              userName = "default";
        }
}
relation nextFragment{
        range Fragment;
        domain Fragment;
        card 1:*;
        must properties{
              userName = "default";
        }
}
relation prevFragment{
        range Fragment;
        domain Fragment;
        card 1:*;
        must properties{
              userName = "default";
        }
}
```

B Entidades de las actividades del ejemplo

```
entity Task:TaskNull@amilab{}


entity Fragment:FragmentNull@amilab{
      properties{

            fragmentType="title";
            fragmentContent="";
      }
}

entity Routine:usarImpresora@amilab{
      relations{
            hasAction=Task:t1@amilab;
            hasAction=Task:t2@amilab;
            hasAction=Task:t3@amilab;
            hasAction=Task:t4@amilab;
            hasAction=Task:t5@amilab;
            hasAction=Task:t6@amilab;
            hasAction=Task:t7@amilab;
            hasAction=Task:t8@amilab;
            hasAction=Task:t9@amilab;
            hasAction=Task:t10@amilab;
            prevAction=Task:TaskNull@amilab;
            nextAction=Task:TaskNull@amilab;
      }
}
```

```
entity Task:t1@amilab{
	relations{

		composesRoutine=Routine:usarImpresora@amilab;
		hasFragment=Fragment:f1_2@amilab;
		hasFragment=Fragment:f1_1@amilab;
		nextAction=Task:t3@amilab;
		prevAction=Task:TaskNull@amilab;
	}
}

entity Task:t3@amilab{
	relations{
		composesRoutine=Routine:usarImpresora@amilab;
		hasFragment=Fragment:f3_1@amilab;
		nextAction=Task:t4@amilab;
		hasFragment=Fragment:f3_2@amilab;
		prevAction=Task:t1@amilab;
	}
}

entity Task:t4@amilab{
	relations{
		nextAction=Task:t5@amilab;
		hasFragment=Fragment:f4_1@amilab;
		composesRoutine=Routine:usarImpresora@amilab;
		prevAction=Task:t3@amilab;
		hasFragment=Fragment:f4_2@amilab;
	}
}

entity Task:t5@amilab{
	relations{
		prevAction=Task:t4@amilab;
		composesRoutine=Routine:usarImpresora@amilab;
		hasFragment=Fragment:f5_1@amilab;
		hasFragment=Fragment:f5_2@amilab;
		nextAction=Task:t6@amilab;
	}
}

entity Task:t6@amilab{
	capabilities{
```

```
            hasToBeRepeated{
                  nTimes=0;
            }
      }
      relations{
            hasFragment=Fragment:f6_2@amilab;
            composesRoutine=Routine:usarImpresora@amilab;
            nextAction=Task:t7@amilab;
            prevAction=Task:t5@amilab;
            hasFragment=Fragment:f6_1@amilab;
      }
}

entity Task:t7@amilab{
      relations{
            composesRoutine=Routine:usarImpresora@amilab;
            hasFragment=Fragment:f7_1@amilab;
            hasFragment=Fragment:f7_2@amilab;
            nextAction=Task:t8@amilab;
            prevAction=Task:t6@amilab;
      }
}

entity Task:t8@amilab{
      relations{
            composesRoutine=Routine:usarImpresora@amilab;
            hasFragment=Fragment:f8_1@amilab;
            hasFragment=Fragment:f8_2@amilab;
            nextAction=Task:t9@amilab;
      }
}

entity Task:t9@amilab{
      relations{
            hasFragment=Fragment:f9_1@amilab;
            prevAction=Task:t8@amilab;
            composesRoutine=Routine:usarImpresora@amilab;
            hasFragment=Fragment:f9_2@amilab;
            nextAction=Task:t10@amilab;
      }
}

entity Task:t10@amilab{
      relations{
```

```
            hasFragment=Fragment:f10_2@amilab;
            composesRoutine=Routine:usarImpresora@amilab;
            hasFragment=Fragment:f10_1@amilab;
            prevAction=Task:t9@amilab;
            nextAction=Task:TaskNull@amilab;
      }
}

entity Fragment:f1_1@amilab{
      properties{
            fragmentType="text";
            fragmentContent="<H1>Pulsar boton de encendido</H1>";
      }
      relations{
            nextFragment=Fragment:f1_2@amilab;
            prevFragment=Fragment:FragmentNull@amilab;
            composesTask=Task:t1@amilab;
      }
}
entity Fragment:f1_2@amilab{
      properties{
      fragmentType="image";
      fragmentContent="encender.png";
      }
      relations{
      composesTask=Task:t1@amilab;
      prevFragment=Fragment:f1_1@amilab;
      nextFragment=Fragment:FragmentNull@amilab;
      }
}

entity Fragment:f3_1@amilab{
      properties{
            fragmentType="text";
            fragmentContent="<H1>Abrir la tapa</H1>";
      }
      relations{
            prevFragment=Fragment:FragmentNull@amilab;
            nextFragment=Fragment:f3_2@amilab;
            composesTask=Task:t3@amilab;
      }
}

entity Fragment:f3_2@amilab{
```

```
        properties{

                fragmentType="image";
                fragmentContent="abrirtapa.png";
        }relations{
                prevFragment=Fragment:f3_1@amilab;
                composesTask=Task:t3@amilab;
                nextFragment=Fragment:FragmentNull@amilab;
        }
}

entity Fragment:f4_1@amilab{
        properties{

                fragmentType="text";
                fragmentContent="<H1>Poner las hojas</H1>";
        }
        relations{
                prevFragment=Fragment:FragmentNull@amilab;
                composesTask=Task:t4@amilab;
                nextFragment=Fragment:f4_2@amilab;
        }
}

entity Fragment:f4_2@amilab{
        properties{
                fragmentType="image";
                fragmentContent="poneroriginales.png";
        }
        relations{
                composesTask=Task:t4@amilab;
                prevFragment=Fragment:f4_1@amilab
                nextFragment=Fragment:FragmentNull@amilab
        }
}

entity Fragment:f5_1@amilab{
        properties{
                fragmentType="text";
                fragmentContent="<H1>Cerrar la tapa</H1>";
        }
        relations{
                composesTask=Task:t5@amilab;
                prevFragment=Fragment:FragmentNull@amilab;
```

```
            nextFragment=Fragment:f5_2@amilab;
        }
}

entity Fragment:f5_2@amilab
{
        properties{
            fragmentType="image";
            fragmentContent="cerrartapa.png";
        }
        relations{
            composesTask=Task:t5@amilab;
            nextFragment=Fragment:FragmentNull@amilab
            prevFragment=Fragment:f5_1@amilab;
        }
}

entity Fragment:f6_1@amilab{
        properties{
            fragmentType="text";
            fragmentContent="<H1>Pulsar boton de copia</H1>";
        }
        relations{
            composesTask=Task:t6@amilab;
            prevFragment=Fragment:FragmentNull@amilab;
            nextFragment=Fragment:f6_2@amilab;
        }
}

entity Fragment:f6_2@amilab{
        properties{
            fragmentType="image";
            fragmentContent="copiar.png";
        }
        relations{
            composesTask=Task:t6@amilab;
            prevFragment=Fragment:f6_1@amilab;
            nextFragment=Fragment:FragmentNull@amilab;
        }
}

entity Fragment:f7_1@amilab{
        properties{
            fragmentType="text";
```

```
            fragmentContent="<H1>Abrir la tapa</H1>";
        }
        relations{
            prevFragment=Fragment:FragmentNull@amilab;
            composesTask=Task:t7@amilab;
            nextFragment=Fragment:f7_2@amilab;
        }
}

entity Fragment:f7_2@amilab{
        properties{
            fragmentType="image";
            fragmentContent="abrirtapa.png";
        }
        relations{
            composesTask=Task:t7@amilab;
            prevFragment=Fragment:f7_1@amilab;
            nextFragment=Fragment:FragmentNull@amilab;
        }
}

entity Fragment:f8_1@amilab{
        properties{

            fragmentType="text";
            fragmentContent="<H1>Recoger las hojas y las copias</H1>";
        }
        relations{
            nextFragment=Fragment:f8_2@amilab;
            prevFragment=Fragment:FragmentNull@amilab;
            composesTask=Task:t8@amilab;
        }
}

entity Fragment:f8_2@amilab{
        properties{
            fragmentType="image";
            fragmentContent="recoger.png";
        }
        relations{
            prevFragment=Fragment:f8_1@amilab;
            composesTask=Task:t8@amilab;
            nextFragment=Fragment:FragmentNull@amilab;
        }
```

```
}

entity Fragment:f9_1@amilab{
	properties{
		fragmentType="text";
		fragmentContent="<H1>Cerrar la tapa</H1>";
	}
	relations{
		nextFragment=Fragment:f9_2@amilab
		prevFragment=Fragment:FragmentNull@amilab
		composesTask=Task:t9@amilab;
	}
}

entity Fragment:f9_2@amilab{
	properties{
		fragmentType="image";
		fragmentContent="encender.png";
	}
	relations{
		prevFragment=Fragment:f9_1@amilab;
		composesTask=Task:t9@amilab;
		nextFragment=Fragment:FragmentNull@amilab;
	}
}

entity Fragment:f10_1@amilab{
	properties{

		fragmentType="text";
		fragmentContent="<H1>Pulsar boton de OFF</H1>";
	}
	relations{
		prevFragment=Fragment:FragmentNull@amilab;
		composesTask=Task:t10@amilab;
		nextFragment=Fragment:f10_2@amilab;
	}
}

entity Fragment:f10_2@amilab{
	properties{

		fragmentType="image";
		fragmentContent="encender.png";
```

```
        }relations{
                composesTask=Task:t10@amilab;
                prevFragment=Fragment:f10_1@amilab;
                nextFragment=Fragment:FragmentNull@amilab;
        }
}
```